MASTER IN COMPUTER SCIENCE

Validation of UMTS emulation through PlanetLab

Evrard, Benjamin; Gomand, Gille

Award date:
2009

Awarding institution:
University of Namur

Facultés Universitaires Notre-Dame de la Paix de Namur
Faculté d'Informatique

Validation of UMTS emulation through PlanetLab

Benjamin EVRARD
Gille GOMAND

Mémoire présenté en vue de l'obtention
du grade de Maître en Informatique

Année académique 2008-2009

Abstract

With UMTS, the 3^{rd} Generation of mobile phones gives to users the opportunity to enjoy applications requiring high data rate (e.g. video streaming). At the University of Namur, Hugues Van Peteghem has designed and implemented a testbed emulating UMTS Release 99 as a part of his PhD Thesis. This Master Thesis aims at validating it by comparing results from the testbed with those from a real environment. The goal is to check the validity of the testbed results (i.e. if it is giving realistic results). This document presents an approach to perform tests and experiments in a real environment from a basic end user perspective as opposed to a supervised experimental set-up. Tests and experiments were both performed from a PlanetLab Node and from a classical Linux-based laptop. The results are not directly comparable to those from the testbed because of discrepancies in the intended environment, the latter being a UMTS Release 5 environment with unknown parameters. However, this dissertation and presented results can be easily used for future work.

Keywords: 3G, UMTS, HSDPA, PlanetLab, testbed.

Résumé

Avec l'UMTS, la 3^{e} génération de téléphones mobiles offre la possibilité aux utilisateurs de profiter d'applications requièrantes un haut débit de données (ex. streaming vidéo). A l'Université de Namur, Hugues Van Peteghem a conçu et implémenté un testbed émulant la version 99 de l'UMTS dans le cadre de sa thèse de doctorat. Ce mémoire vise à le valider en comparant les résultats venant du testbed avec ceux venant d'un environnement réel. L'objectif est de vérifier la validité des résultats du testbed (càd, s'il donne des résultats réalistes). Ce document présente une approche pour réaliser des tests et des expériences dans un environnement réel selon la perspective d'un utilisateur final de base par opposition à une installation expérimentale contrôlée. Les tests et les expériences ont été réalisés à partir d'un noeud PlanetLab, ainsi qu'à partir d'un ordinateur portable classique fonctionnant sous Linux. Les résultats ne sont pas directement comparables avec ceux venant du testbed à cause de divergences dans l'environnement attendu, ce dernier étant un environnement UMTS de version 5 avec des paramètres inconnus. Cependant, ce mémoire et les résultats présentés peuvent être facilement utilisés pour de futurs travaux.

Mots clés: 3G, UMTS, HSDPA, PlanetLab, testbed.

Acknowledgements

We would like to express our gratitude to all of those who gave us the possibility to complete this Master Thesis.

Professor Roberto Canonico, our internship supervisor,
for his availability, support, and advices concerning our work during our stay at the CINI laboratory of the University of Naples Federico II in Italy. We also would like to thank the *CINI laboratory team* to have cordially received us. In particular, *Giovanni Di Stasi* for his constant support and his help during our first days in Napoli and *Alessio Botta* for his valuable advices.

Professor Laurent Schumacher, our Thesis supervisor,
for his high availability and precious advices as well as his feedback and his many corrections.

Hugues Van Peteghem, PhD, and *George Toma*, researcher at the University of Namur,
for their help and advices relating to some parts of our Thesis.

Our families and friends,
for their understanding and their continuous support during our studies.

Contents

List of Figures

List of Tables

List of Acronyms

1G	1^{st} Generation
2G	2^{nd} Generation
3G	3^{rd} Generation
3GPP	3^{rd} Generation Partnership Project
4G	4^{th} Generation
ACK	Acknowledgement
AM	Acknowledged Mode
AMC	Adaptive Modulation and Coding
AMR	Adaptive MultiRate
APN	Access Point Name
ARQ	Automatic Repeat reQuest
ASCII	American Standard Code for Information Interchange
BER	Bit Error Rate
CAC	Call Admission Control
CDMA	Code Division Multiple Access
CINI	Consorzio Interuniversitario Nazionale per l'Informatica
CN	Core Network
CS	Circuit-Switched
DCH	Dedicated CHannel
DoS	Denial of Service
DS-CDMA	Direct-Sequence Code Division Multiple Access
DSCH	Downlink Shared CHannel
DSL	Digital Subscriber Line
E2E	End-to-End

EDGE	Enhanced Data rates for GSM Evolution
ETSI	European Telecommunications Standards Institute
FACH	Forward Access CHannel
FDM	Frequency Division Multiplex
FDMA	Frequency Division Multiple Access
GGSN	Gateway GPRS support node
GoP	Group of Pictures
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communications
HARQ	Hybrid ARQ
HS-DSCH	High-Speed Downlink-Shared Channel
HSDPA	High Speed Downlink Packet Access
HSPA	High Speed Packet Access
HSUPA	High Speed Uplink Packet Access
IP	Internet Protocol
IPv6	IP version 6
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider
IVR	Interactive Voice Response
LB	Linux Box
LED	Light-Emitting Diode
LTE	Long-Term Evolution
MPEG-1	Motion Picture Experts Group Layer-1
MPEG-4	Motion Picture Experts Group Layer-4
MT	Mobile Terminal
NAT	Network Address Translation
NIC	Network Interface Card
NodeB	WCDMA base station
NT	Namur Testbed
OVSF	Orthogonal Variable Spreading Factor
PCMCIA	Personal Computer Memory Card International Association

PDF	Portable Document Format
PDP	Packet Data Procotol
PDU	Protocol Data Unit
PEAQ	Perceptual Evaluation of Audio Quality
PESQ	Perceptual Evaluation of Speech Quality
PIN	Personal Identification Number
PL	PlanetLab
PLC	PlanetLab Central
PLE	PlanetLab Europe
PLMN	Public Land Mobile Network
PLN	PlanetLab Node
PPP	Point-to-Point Protocol
PRN	PrivateLab Node
PS	Packet-Switched
PSNR	Peak Signal-to-Noise Ratio
PSTN	Public Switched Telephone Network
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RDT	Real Data Transport
Rel'5	UMTS Release 5
Rel'6	UMTS Release 6
Rel'7	UMTS Release 7
Rel'8	UMTS Release 8
Rel'99	UMTS Release 99
RLC	Radio Link Control
RNC	Radio Network Controller
RRM	Radio Resource Management
RSSI	Received Signal Strength Indication
RT	Real-Time
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
RTSP	Real-Time Streaming Protocol
RTT	Round-Trip Time
SDU	Service Data Unit
SF	Spreading Factor
SIM	Subscriber Identity Module
SIP	Session Initiation Protocol
SMS	Short Message Service
SNodeB	Serving NodeB
STUN	Session Traversal Utilities for NAT

TCP	Transport Control Protocol
TDM	Time Division Multiplex
TDMA	Time Division Multiple Access
TEMS	TEst Mobile System
TG	Traffic Generator
TM	Transparent Mode
TTI	Transmission Time Interval
UDP	User Datagram Protocol
UE	User Equipment
UM	Unacknowledged Mode
UMTS	Universal Mobile Telecommunications System
UPMC	Université Pierre et Marie Curie
URL	Uniform Resource Locator
UTRAN	UMTS Terrestrial RAN
VoIP	Voice over IP
WCDMA	Wideband CDMA
XML	Extensible Markup Language

Chapter 1

Introduction

Since the introduction of the Global System for Mobile communications (GSM), also called 2^{nd} Generation (2G), the mobile phones have constantly increased in popularity. Nowadays, the number of mobile phones exceed the number of landline phones. In many markets, the mobile phone penetration is over 100%. The introduction of the Universal Mobile Telecommunications System (UMTS), also called 3^{rd} Generation (3G), which is now widely widespread, provides the high bit-rate services that enable high-quality images and video to be transmitted and received, and to provide access to the Web with higher data rates. Users have now the opportunity to enjoy multimedia applications on their mobile phones. The iPhone from Apple is the perfect illustration of this new generation of mobile phones that can take advantage of the 3G providing high data rate.

At the University of Namur, as a part of his PhD Thesis [1], Hugues Van Peteghem has designed and implemented a testbed which is representing the radio part of the UMTS, called UMTS Terrestrial RAN (UTRAN). This testbed will be called the Namur Testbed (NT) in the following. His objective was "to establish a suitable open-source software/hardware Real-Time (RT) platform in order to thoroughly analyse and improve Quality of Service (QoS) management in the UTRAN." [1]

The NT gives to operators and infrastructure manufacturers the possibility to test and fine tune their existing algorithms or new algorithms based on specific scenarios. This is a very interesting tool because of its very low cost and its easiness as compared with testing in a real environment.

The issue is whether such a testbed emulating a real environment is giving realistic results, or at least as realistic as possible. This is the topic of this Master Thesis. Originally, the main objective was to reproduce experiments, which were performed on the NT, in a real environment in order to compare results with those of the NT. The goal is to validate it by checking the validity of its results (i.e. if it is giving realistic results).

However, as explained in our traineeship report [2], because of discrepancies in the intended environment, the results to be presented in this document will not be directly comparable with those of the NT. Therefore, this document will rather present an approach to perform tests and experiments in a real environment from a basic end user perspective, where network parameters are uncontrolled and unknown, as opposed to a supervised experimental set-up, where the network is under control. Besides being uncontrolled, the real environment in which

experiments were performed is a UMTS Release 5 (Rel'5) environment, better known as High Speed Downlink Packet Access (HSDPA), whereas the NT complies with UMTS Release 99 (Rel'99). Nevertheless, this dissertation and presented results can be easily used for future works. All the more so the NT will evolve to UMTS Release 5.

The Consorzio Interuniversitario Nazionale per l'Informatica (CINI) laboratory in Napoli, where our traineeship and the experiments were carried out, is registered in OneLab and PlanetLab consortiums. Being a worldwide consortium, the PlanetLab network is assimilated to a Virtual Internet. The members of such consortia provide powerful computers used by researchers to perform their experiments and tests in realistic deployment environment.

The CINI laboratory is responsible of the deployment and the integration of the UMTS connection on a PlanetLab Node. The tests and experiments described in this dissertation were both performed from a PlanetLab environment (using a PlanetLab Node) and from a classical environment (using a common Linux-based laptop), in order to compare results from different environment.

The present dissertation is structured as follows:

Chapter 2 describes the main parts of the UMTS universe and its mechanisms needed to fully understand the studied aspects in this document. It also introduces the evolution of the UMTS Release 99, called HSDPA (UMTS Release 5).

Chapter 3 presents the Namur Testbed, its architecture, characteristics and limitations which are related to the scope of our work.

Chapter 4 explains the PlanetLab environment which was a part of the experiments environment.

Chapter 5 provides a brief state of the art.

Chapter 6 aims to explain the real environment in which the experiments were performed. It describes the hardware, and the tools used to connect to UMTS.

Chapter 7 explains the tests realized to (try to) discover some parameters of the cellular network used for our experiments.

Chapter 8 describes the experiments, tools used, and their configurations. Video streaming and Voice over IP (VoIP) experiments were both performed using real and synthetic traffic.

Chapter 9 analyses the results obtained by running experiments in terms of QoS metrics (jitter and packet loss rate) computed at the application layer. For the video streaming experiments using real traffic, Quality of Experience (QoE) evaluation is performed by using Peak Signal-to-Noise Ratio (PSNR) metric.

Chapter 10 concludes and presents some future work outlooks.

Chapter 2

UMTS

This chapter introduces the main concepts from the UMTS universe including the key elements of the UMTS and HSDPA technologies needed to fully understand the studied aspects in this document. The descriptions are keeping as simply as possible. Some detailed explanations of the UMTS technology are beyond the scope of this document. Reader familiar with UMTS can safely skip this chapter.

2.1 UMTS architecture overview

As shown in Fig. 2.1, UMTS systems can be roughly divided into three (network) parts: the air interface, the Radio Access Network (RAN), and the Core Network (CN).

The air interface is the technology of the wireless link located between the NodeB and the User Equipment (UE), which interfaces with the user. The Core Network is responsible for switching and routing calls and data connections to external networks, giving access to the wider Internet or Public Switched Telephone Network (PSTN). The Radio Access Network handles all radio-related functionality (e.g., dealing with most of the consequences of the terminal's mobility, etc.).

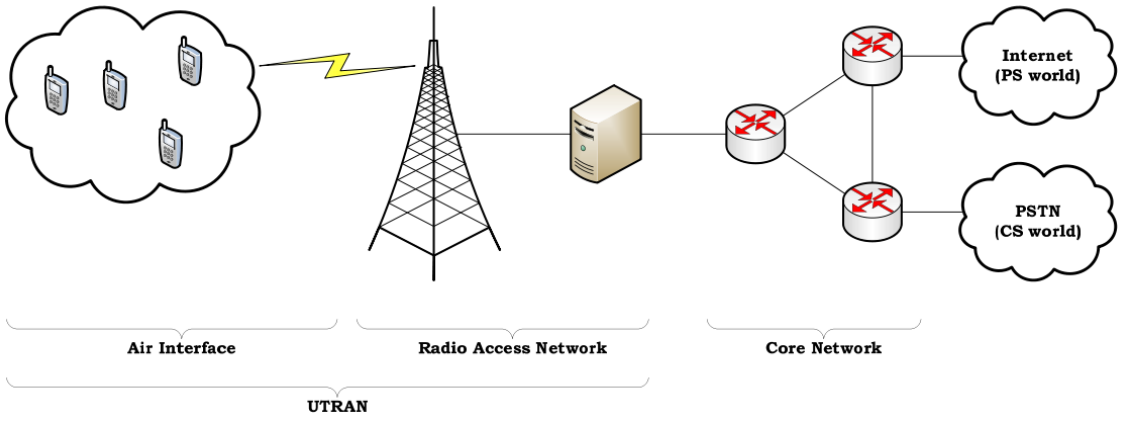


Figure 2.1: Third generation network overview. [1]

Another way to group UMTS network elements is to divide them into sub-networks. The UMTS is modular and can have several network elements of the same type. This allows the division of the UMTS into sub-networks. The latter can operate either on their own or together with other sub-networks. Such a sub-network is called an UMTS Public Land Mobile Network (PLMN). Typically, one PLMN is operated by a single operator, and is connected to other PLMNs as well as to other types of network, such as ISDN, PSTN, the internet, and so on. Fig. 2.2 gives a representation of the elements in a PLMN, connected to external networks.

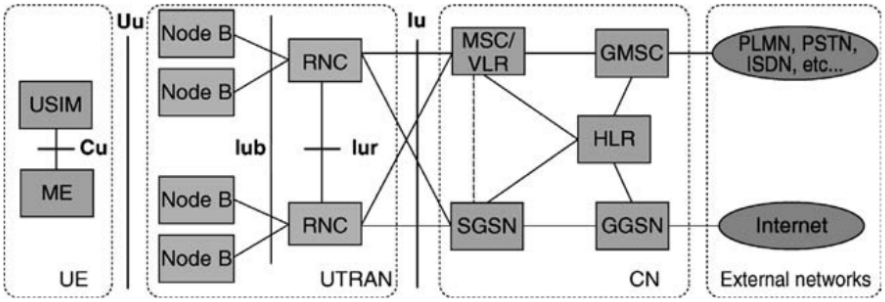


Figure 2.2: Network elements in a PLMN. [3]

Hereunder is given a short introduction to all the elements [3].

The *UE* consists of two parts:

- The *Mobile Equipment (ME)* is the radio terminal used for radio communication over the Uu interface.
- The *UMTS Subscriber Identity Module (USIM)* is a smartcard that holds the subscriber identity, performs authentication algorithms, and stores authentication and encryption keys and some subscription information that is needed at the terminal.

UTRAN also consists of two distinct elements:

- The *NodeB* converts the data flow between the Iub and Uu interfaces. It also participates in radio resource management.
- The *Radio Network Controller (RNC)* owns and controls the radio resources in its domain (the NodeBs connected to it). The RNC is the service access point for all services that UTRAN provides the CN, e.g. management of connections to the UE.

The following *CN* elements are only briefly described:

- *Home Location Register (HLR)* is a database located in the user's home system that stores the master copy of the user's service profile.
- *Mobile Services Switching Centre/Visitor Location Register (MSC/VLR)* is the switch (MSC) and database (VLR) that serves the UE in its current location for Circuit-Switched (CS) services. The MSC function is used to switch the CS transactions, and the VLR function holds a copy of the visiting user's service profile, as well as more precise information on the UE's location within the serving system.
- *Gateway MSC (GMSC)* is the switch at the point where UMTS PLMN is connected to external CS networks. All incoming and outgoing CS connections go through GMSC.

- *Serving GPRS Support Node (SGSN)* functionality is similar to that of MSC/VLR but is typically used for Packet-Switched (PS) services.
- *Gateway GPRS Support Node (GGSN)* functionality is close to that of GMSC but is in relation to PS services.

The *external networks* can be divided into two groups:

- *CS networks* provide Circuit-Switched connections, like the existing telephony service. ISDN and PSTN are examples of CS networks.
- *PS networks* provide connections for packet data services. The internet is one example of a PS network.

The interfaces are the following:

- *Cu interface* is the electrical interface between the USIM smartcard and the ME.
- *Uu interface* is the Wideband CDMA (WCDMA) radio interface. The Uu is the interface through which the UE accesses the fixed part of the system.
- *Iu interface* connects UTRAN to the CN.
- *Iur interface* allows soft handover between RNCs from different manufacturers.
- *Iub interface* connects a NodeB and a RNC.

2.2 Multiple access methods

In computer networks and telecommunications, a multiple access method (or channel access method) allows several terminals connected to the same multi-point transmission medium to transmit over it and to share its capacity. A channel access scheme is based on a multiplex method, which allows several data streams or signals to share the same communication channel or physical media.

These are different fundamental forms of channel access schemes :

FDMA

Frequency Division Multiple Access (FDMA) is a channel access method based on the Frequency Division Multiplex (FDM) scheme, which provides different frequency bands to different data streams (users or nodes). FDMA gives users an individual allocation of one or several frequency bands, or channels.

Early systems, now referred to as 1^{st} Generation (1G), used this analog technology, called FDMA, to deliver a radio-based voice channel to a mobile telephone user.

TDMA

Time Division Multiple Access (TDMA) is a channel access method based on the Time Division Multiplex (TDM) scheme which shares the same frequency by dividing the signal into different time slots. Therefore, it provides different time slots to different transmitters in a cyclically repetitive frame structure.

In the late 1980s, 2^{nd} Generation systems were deployed using digital technologies. The first U.S. system used TDMA. In the early 1990s, this technology was used to introduce the GSM to Europe. Actually, the GSM uses combined FDM/TDM for the air interface.

CDMA

Code Division Multiple Access (CDMA) employs a special coding scheme. CDMA assigns a specific, orthogonal code to each node. Each node then uses its unique code to encode the data bits it sends. Therefore, different nodes are multiplexed and can transmit simultaneously over the same physical channel.

In case of wireless physical channel, all users share the same radio frequency at the same time. Each user in a cell is allocated a distinct sequence of bits, called a chipping sequence. Each bit being sent is encoded by multiplying the bit by a signal (the code) that changes at a much faster rate (known as the chipping rate) than the original sequence of data bits.

In the mid 1990s, CDMA became the second type of digital 2G system, with the U.S. introduction of Interim Standard-95 (IS-95), now referred to as cdmaOne.

Fig. 2.3 shows a representation of these different multiple access methods.

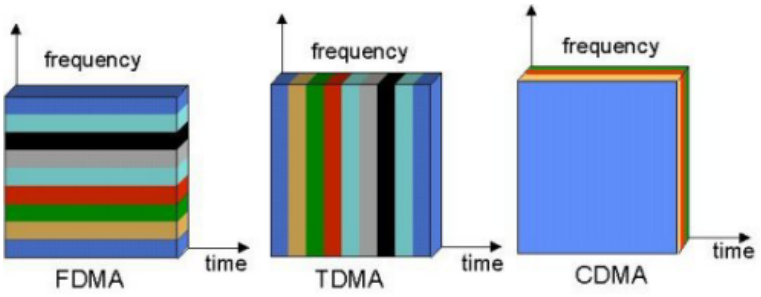


Figure 2.3: Allocation of bandwidth in FDMA, TDMA and CDMA. [4]

2.3 WCDMA

While the IS-95 (or cdmaOne) evolved to CDMA2000 in 3G, the UMTS, which is an evolution of GSM to support 3G capabilities, uses the WCDMA technology.

WCDMA is a wideband Direct-Sequence Code Division Multiple Access (DS-CDMA) system. The user data are spread over a wide bandwidth by multiplying them with quasi-random bits (called chips) derived from CDMA spreading codes. A variable Spreading Factor (SF) and multicode connections are used to support very high bit rates (up to 2 Mbps). An example of this arrangement is shown in Fig. 2.4.

It is good to point out that handovers between GSM and WCDMA are supported in order to be able to leverage the GSM coverage for the introduction of WCDMA.

As already explained, CDMA uses unique spreading codes to spread the data before transmission. The receiver then uses a correlator to despread the wanted signal, without being annoyed by unwanted signals. The rate of a spreading code, referred to as chip rate rather than bit rate, is produced at a much higher rate than the one of the data rate.

WCDMA uses Direct-Sequence spreading, where spreading process is done by directly combining the baseband information to high chip rate binary code. The Spreading Factor is the ratio of the chip to baseband information rate.

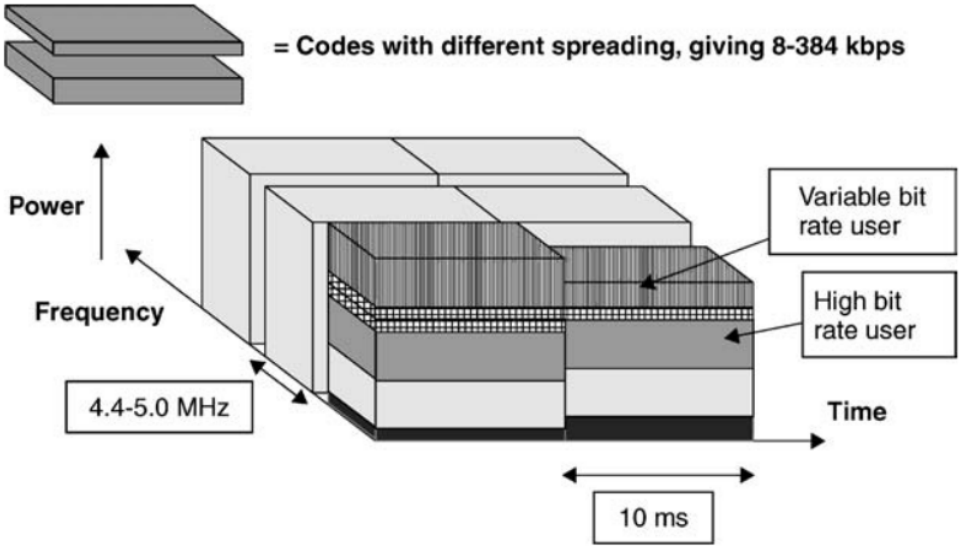


Figure 2.4: Allocation of bandwidth in WCDMA in the time–frequency–code space. [3]

Since the UMTS uses WCDMA as air interface, it respects the spreading/despreading concept [5]. The spreading/channelisation codes of UTRAN are based on the Orthogonal Variable Spreading Factor (OVSF) technique.

By using OVSF codes, the Spreading Factor can be changed and the orthogonality can be maintained between different spreading codes of different lengths. The codes are picked from the code tree, which is illustrated in Fig. 2.5. When a connection uses a variable Spreading Factor, it is allowed to despread according to the smallest Spreading Factor. The only requirement is that channelisation codes are used from the branch indicated by the code used for the smallest Spreading Factor. The downlink orthogonal codes within each base station are managed by the RNC in the network.

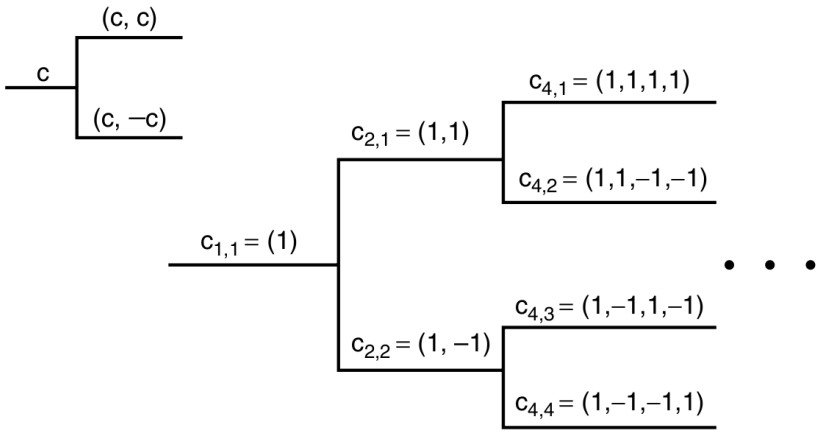


Figure 2.5: Beginning of the channelisation code tree. [3]

When a particular Spreading Factor is allocated, e.g. Spreading Factor 8, then the codes as part of the same branch (sub-tree) cannot be used anymore. Therefore, from the code tree, booking a Spreading Factor 8 will occupy $1/8^{th}$ of the total code tree resource (Fig. 2.6).

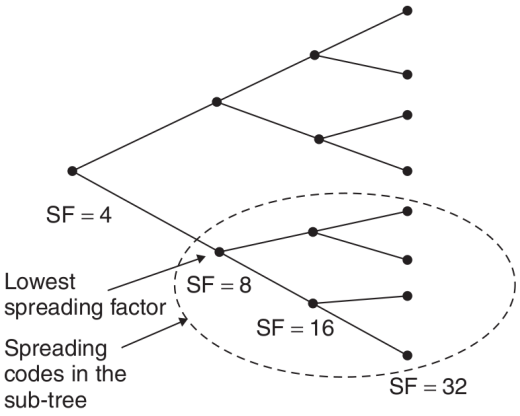


Figure 2.6: Code tree example. [3]

Note there is a difference between uplink and downlink. In the downlink direction the code tree is shared by the users of the cell or sector; in the uplink, all users have independent code trees.

The spreading code allocated and the bandwidth are linked since the SF defines how many chips are used to code one user data symbol. Therefore, higher SFs will subsequently reduce the data rate. WCDMA supports a maximal chip rate of 3.84 Mchips/s and the available downlink SFs vary from 4 to 512.

2.4 Transport Channels

In the UTRAN, data generated at higher layers are carried over the air interface using transport channels. All these channels are defined as unidirectional (i.e. uplink or downlink).

The are two types of transport channel: Dedicated CHannels (DCHs) and common channels. The main difference between them is that a common channel is a resource divided between all or a group of users in a cell, whereas a DCH resource (identified by a certain code on a certain frequency) is reserved for a single user only.

The only dedicated transport channel is the Dedicated CHannel (DCH). It carries the service data and control information of a single user at variable bit rate. The DCH is characterised by features such as fast power control and fast data rate change on a frame-by-frame basis.

There are six different common transport channel types defined for the Rel'99 UTRAN. We only focus on two of them that are described in the PhD Thesis [1] and implemented in the NT: "The Downlink Shared CHannel (DSCH) which carries signalling and service data of several users at variable bit rates, and the Forward Access CHannel (FACH) which carries also the service data of several users but at a fixed bit rate." [1]

As presented in Table 2.1, the FACH and DCH carry user data flow with a fixed bit rate depending on their SF. The DSCH has a variable bandwidth since its SF occupies the free room left in the OVSF tree after the DCHs SF allocation.

Table 2.1: Comparison of different transport channels.

Channel	**DCH**	**DSCH**	**FACH**
SF	Fixed [512-4]	Variable [256-4]	Fixed [256-4]
Multiusers	Dedicated	Shared	Shared

2.5 RLC modes

The Radio Link Control (RLC) protocol provides segmentation and retransmission services for both user and control data. The UMTS RLC layer is able to transmit RLC Protocol Data Units (PDUs) in three different modes: Transparent Mode (TM), Unacknowledged Mode (UM) and Acknowledged Mode (AM).

Hereunder is a brief description of these modes.

Transparent Mode

The TM is the simplest mode, no protocol overhead is added to higher layer data. A TM RLC entity does not add any RLC protocol overhead to Service Data Units (SDUs). The entity is transparent to SDUs.

Unacknowledged Mode

In UM, no retransmission protocol is in use. There is no guarantee of correct delivering of RLC PDUs to the peer entity. However, RLC overheads are adding to SDUs. The header provides a sequence number. Since RLC PDUs are numbered, a UM entity can provide segmentation of SDUs and their reassembling at the receiver side. If a missing RLC PDU is detected by the receiving UM entity, the latter automatically discards all RLC PDUs that belong to the same SDU.

Acknowledged Mode

AM is the most elaborated RLC mode. This mode is used if a reliable data transfer over the UMTS radio interface is required. An AM RLC can deliver reassembled SDUs to the upper layer either in sequence or out of sequence. This configuration of the RLC uses the Automatic Repeat reQuest (ARQ) technique. It provides reliable data transfers and handles errors by retransmitting data according to receiver demands. This technique requires one bi-directional connection between the transmitter and the receiver, i.e. the UE and the NodeB. The direction from the transmitter to the receiver is used for data transmission. The opposite direction is used by the receiver to transfer status reports. They allow the receiver to inform the transmitter about the RLC PDUs which have to be retransmitted.

2.6 QoS traffic classes

3^{rd} Generation Partnership Project (3GPP) defined four QoS traffic classes : Conversational, Streaming, Interactive and Background. The representative applications are, respectively, VoIP, web browsing, video streaming and e-mail. Each of these classes have some End-to-End (E2E) QoS requirements to be met by the network in order to function properly. For example,

e-mail applications (Background) have a better tolerance to delays than VoIP applications (Conversational).

2.7 HSDPA

High Speed Downlink Packet Access (HSDPA) is an upgrade on Rel'99 of the UMTS standard. This is a new standard, defined as UMTS Release 5, that extends WCDMA to increase the downlink communication rate and reduce the latency of the link. It enables a data speed of approximately (maximum) 14 Mbps. The Round-Trip Time (RTT) can be pushed below 70 ms while the RTT of a WCDMA UMTS Release 99-dedicated channel is typically 100–200 ms.

HSDPA is a protocol of the High Speed Packet Access (HSPA) family, positioned as a 3.5G service. It functions as a bridge between WCDMA, generally known as 3G services, and the 4^{th} Generation (4G) service slated for future introduction.

In HSDPA, the variable SF and the fast power control are disabled and replaced by means of Adaptive Modulation and Coding (AMC), extensive multi-code operation and a fast and spectrally efficient retransmission strategy. The main characteristics of HSDPA are described hereunder.

Hybrid ARQ (HARQ) improves on the conventional ARQ that is essential to wireless communications under constantly varying conditions. The conventional ARQ requests repeatedly for a packet data with a decoding quality when an error is found. Differently from this, HARQ combines retransmitted data with received data that was not yet decoded in order to improve the reception quality and achieve a more efficient transmission.

Adaptive Modulation and Coding Scheme assesses the status of continuously varying radio wave propagation paths while maintaining a constant transmission power and then automatically selects the optimal modulation method. To maintain a good spectral efficiency and to enable a large dynamic range of the HSDPA link adaptation, a user may simultaneously utilise up to 15 multi-codes in parallel. By using a more robust coding, a fast HARQ and a multi-code operation, a variable SF is not needed anymore. The SF is fixed and is always 16. The total number of channelisation codes with SF 16 is respectively 16. However, as a code space for common channels is needed, the maximum usable number of codes is set to 15.

High-Speed Downlink-Shared Channel (HS-DSCH) is the transport channel carrying the user data with HSDPA operation. This channel is dynamically time- and code-multiplexed and allocated to multiple users' data transmissions, resulting in more efficient allocation of wireless resources. The idea in HSDPA is to enable a scheduling such that, if desired, most of the cell capacity may be allocated to one user for a very short time, when conditions are favourable. The Transmission Time Interval (TTI) (or interleaving period) has been defined to be 2 ms to achieve a short round-trip delay for the operation between the terminal and NodeB for retransmissions. Therefore, for a given 2 ms frame, data may be sent to a number of users simultaneously, using different channelisation codes. The maximum number of users to receive data on a given 2 ms frame is determined by the number of allocated channelisation codes.

2.8 Evolution

WCDMA was the first air interface adopted by the European Telecommunications Standards Institute (ETSI) for the 3G, called UMTS Release 99 (Rel'99).

Later, 3GPP specified an important evolution on top of WCDMA, called HSPA, composed of two major steps: High Speed Downlink Packet Access (HSDPA), called UMTS Release 5 (Rel'5) and High Speed Uplink Packet Access (HSUPA), called UMTS Release 6 (Rel'6). Further HSPA evolution is specified in 3GPP UMTS Release 7 (Rel'7), also known as HSPA+.

In the same time, 3GPP is also working on a new radio system called Long-Term Evolution (LTE). The standard was frozen in December 2008. UMTS Release 7 (Rel'7) and UMTS Release 8 (Rel'8) solutions for HSPA evolution will be worked in parallel with LTE development, and some aspects of LTE work are also expected to reflect on HSPA evolution.

Fig. 2.7 shows the peak data rate evolution through the different releases.

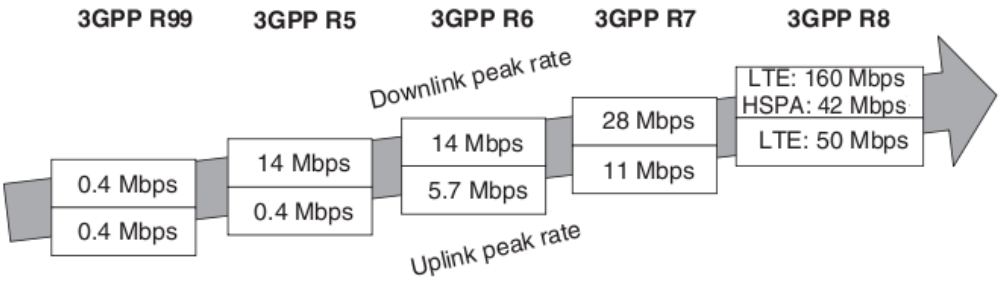


Figure 2.7: Peak data rate evolution for WCDMA. [3]

Chapter 3

The Emulating UTRAN Testbed

This chapter gives a short description of the Namur Testbed that emulates a UMTS Release 99 access network (the UTRAN part). This testbed was realized by Hugues Van Peteghem as part of his PhD Thesis at the University of Namur that he handed in March 2008. The following description is based on some excerpts from his PhD Thesis, "Building a Testbed Emulating Cellular Networks; Design, Implementation, Cross-Validation and Exploitation of a Real-Time Framework to Evaluate QoS and QoE in the UTRAN" [1]. This chapter focuses on the characteristics of the testbed that are related to the scope of our work. For further details, please refer to the Thesis [1].

3.1 Testbed goals

As described in [1], to develop and test new wireless technologies, different methodologies are available and can be classified in four tracks.

The first track uses analytical developments. Thanks to network calculus tools, statistical modelling and finite state machines, it is possible to create a model of the behaviour of computer networks.

The second track is the simulation type. Due to the complexity of communication networks, it is difficult to have a good understanding of them while using only analytical studies. Networks simulators allow to test scenarios that might be difficult or expensive to emulate using real hardware. For example, simulating the effects of a sudden burst in traffic or a DoS attack on a network service.

The third track is the measurement campaigns on a testbed. When the size and the complexity of a network simulation increase, networks simulators are limited due to scalability issues and implementation difficulties. In this case, testbeds can be considered as a complementary tool. Besides being cheaper than a real deployment, they emulate real network behaviour in a more accurate way than a simulator would do in complex world simulations (due to hardware limitations). Moreover, testbeds are more suitable to reproduce the behaviour of a single user while simulators are more suitable for global systems performance analysis.

The fourth track combines all the field tests. While testbeds are based on assumptions that could influence and bias the conclusion made, the advantage of field tests is that results are

directly applicable without any post-processing. However, this approach is more expensive and has reproducibility issue of test environment when confronting different results.

The approach of Hugues Van Peteghem's Thesis is a third track approach while using the other tracks to cross-validate the obtained results. The objective was "to establish a suitable open-source software/hardware RT platform in order to thoroughly analyse and improve QoS management in the UTRAN." [1]

In contrast with most testbeds, that are restricted to a single service, the NT is open to all typical UMTS applications. "It shall work on an acceptable QoS level for all these simultaneously active services e.g. a population of users within the same cell, some of them having standard conversations (Conversational), others browsing the Internet (Interactive) or retrieving their Emails (Background), while a few more users arrive and initiate a video streaming session (Streaming)." [1]

In addition to the possibility to test E2E QoS performance (metrics like packet delay, loss rate, jitter, cell throughput, etc.), the NT is able to assess the QoE perceived by the end user in many different scenarios (mobility, traffic load, traffic type, etc.).

These advantages give operators and infrastructure manufacturers the possibility to test and fine tune their existing algorithms or new algorithms based on specific scenarios.

3.2 Testbed architecture

As shown in Fig. 3.1, the NT "emulates a Rel'99 UTRAN segment consisting of one RNC managing four NodeBs, each of them serving a population of UEs. The NodeBs are placed on a hexagonal grid being the standard macrocellular set-up [6, 7]." [1]

The NT, that emulates this UTRAN segment, is composed of nine Linux-operated personal computers. Linux Red-Hat-based Fedora Core 6, kernel 2.6.20 was chosen as operating system due to its flexibility, its ability of being customized, and its cost/performance ratio. Computers are interconnected with 100 Mbit/s Ethernet links, creating a little network working over IP version 6 (IPv6) and isolated from the Internet. From the implementation perspective, this testbed is fully developed in the C language and can be seen as an assembly of numerous modules.

The network is subdivided in three computer groups (A, B and C).

Group A - NodeBs

"The computers in group A represent four independent trisectorial macrocell NodeBs. Their role is to take care of the traffic coming in from the UE (resp. RNC) and to forward it to the RNC (resp. UE). Most of the QoS management functions are performed at these spots as they are the bottlenecks of the UTRAN (border between the wireless and the wired networks). Since the network does not contain any real NodeB, these computers handle the emulation of the UMTS air interface (BER, Service Data Unit (SDU) segmentation), etc.).

In the early UMTS standardisation the NodeBs were seen as simple access points linking the UEs to the UTRAN and the CN. With the introduction of HSDPA, they gained more responsibilities in terms of QoS management and packet scheduling. Therefore, in anticipation

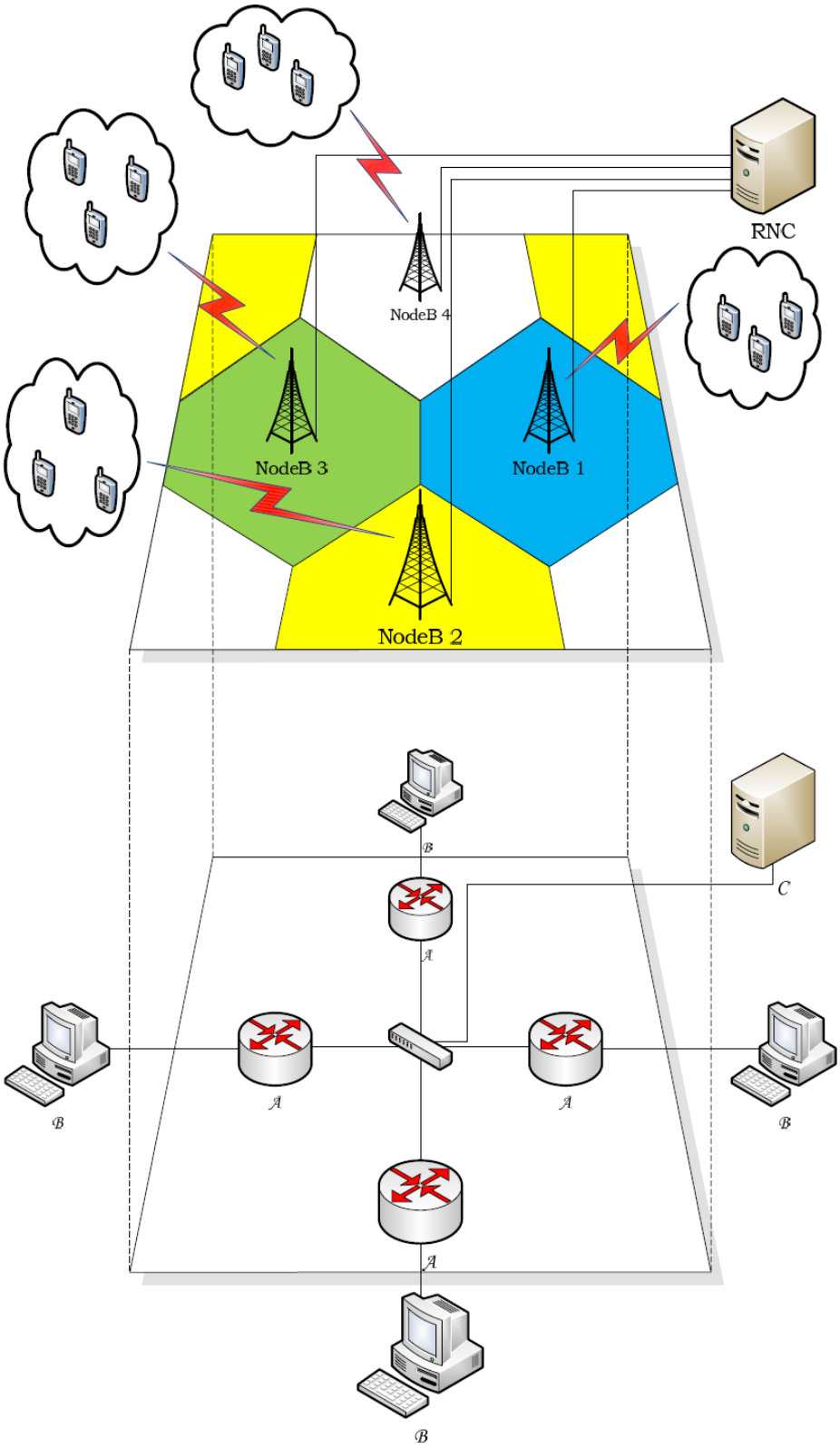


Figure 3.1: UTRAN testbed representation. [1]

of a future evolution of the testbed, the computers emulating the NodeBs are designed in a Rel'5 philosophy while the testbed is still emulating a Rel'99 UTRAN." [1]

Group B - UEs

"Each computers of this group act as a population of UEs attached to their Serving NodeB (SNodeB) (emulated on the attached A group computer). Each of them generates several uplink flows (from the UE to the RNC) representing the emulated UEs network activity.

In order to simultaneously emulate several autonomous UEs on a single computer, these computers use a control panel based on virtual network interfaces management. A physical network interface can have multiple IP addresses assigned to it." [1]

Group C - RNC

This group is only composed of one computer acting as the testbed's unique RNC. "It is the sink of the traffic coming from the four NodeBs and generates the downlink traffic flows (from the RNC to the UEs) for each of the emulated UEs. As in a real UTRAN, the complexity is gathered in the RNC since it is in charge of the CAC and RRM. It consequently controls the entire emulation." [1]

3.3 Testbed characteristics and limitations

This section describes the characteristics and limitations of the NT which are related to the scope of our tests and experiments in real environment.

3.3.1 RAN & Air Interface

As there is no radio transmissions, the lower layers of the UMTS air interface are emulated over a classical Ethernet (wired) link as shown in Fig. 3.2, as well as the properties of wireless networks. The most important features that characterize these networks are a limited bandwidth, more errors encountering than wired networks, and these errors are time-correlated.

The limited bandwidth availability over an air interface is emulated thanks to the Token Bucket traffic shaper and rate limiter. The Token Bucket controls the amount of data that is injected into a network.

The Traffic Controller Network Emulation (NetEm)[1] is used to emulate the complex behaviour of the variable BER property. Indeed, this property is "slightly more complicated to emulate, since it has to take into account a larger number of parameters such as user speed, its position, the environment in which it is evolving, etc." [1]

Also, "the RAN RTT (computed between the RNC and the UE) is much more important in a real UTRAN since the packets are evolving on a wired network. In order to be as close to reality as possible, the testbed includes a fixed delay to each packet transiting over the emulated layers. This delay has been fixed to 50.2ms following the 3GPP specifications [8].

[1]NetEm is a waiting queue capable to delay, drop, corrupt, duplicate or even record packets.

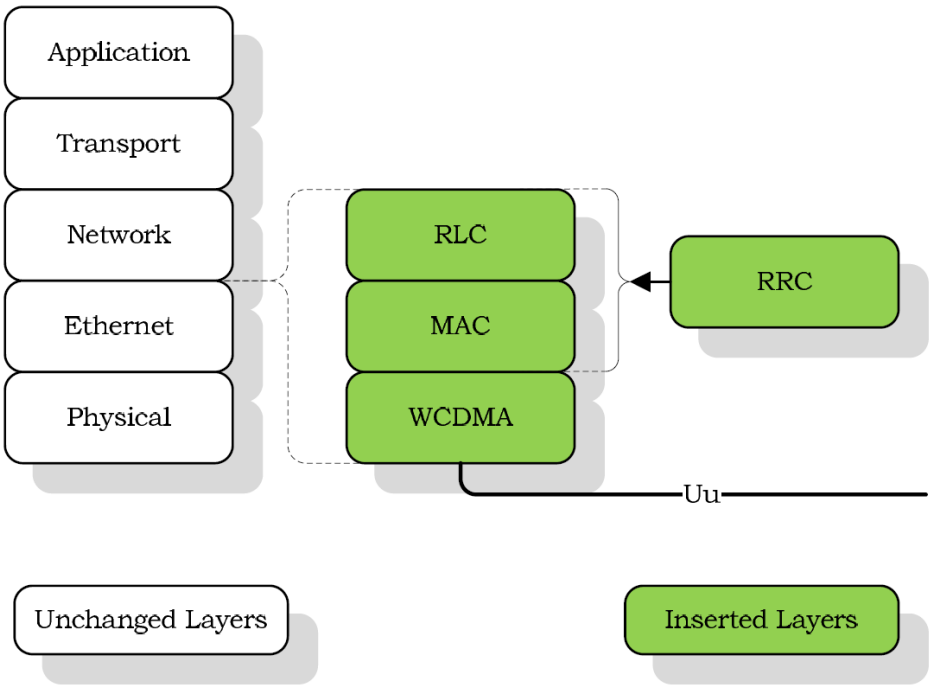


Figure 3.2: UMTS lower layer emulation. [1]

An other assumption which had been set due to the air interface emulation is the DCH allocation time. This is the delay a UE undergoes between its DCH request and the time point it transmits data through it. This allocation time mainly contains the Call Admission Control (CAC) procedure made by the RNC to allocate a SF to the UE. This delay duration has been fixed to 900ms as proposed in [9].” [1]

Finally, the NT focuses only on two of the RLC modes: Transparent Mode (TM) and Acknowledged Mode (AM); this choice has been determinated by the kind of traffic generation.

3.3.2 Downlink-Oriented

“The testbed is focused on the downlink. The major part of the data is transported in this direction. Therefore, the air interface emulation influences only on this direction. For example, considering an Interactive session, the testbed affects, with a given Bit Error Rate (BER) and bandwidth restriction, the Transport Control Protocol (TCP) segments used to download a web page, but the Acknowledgement (ACK) segments returned by the UE do not suffer any additional perturbation or bandwidth restriction.”[1]

3.3.3 Traffic Generation

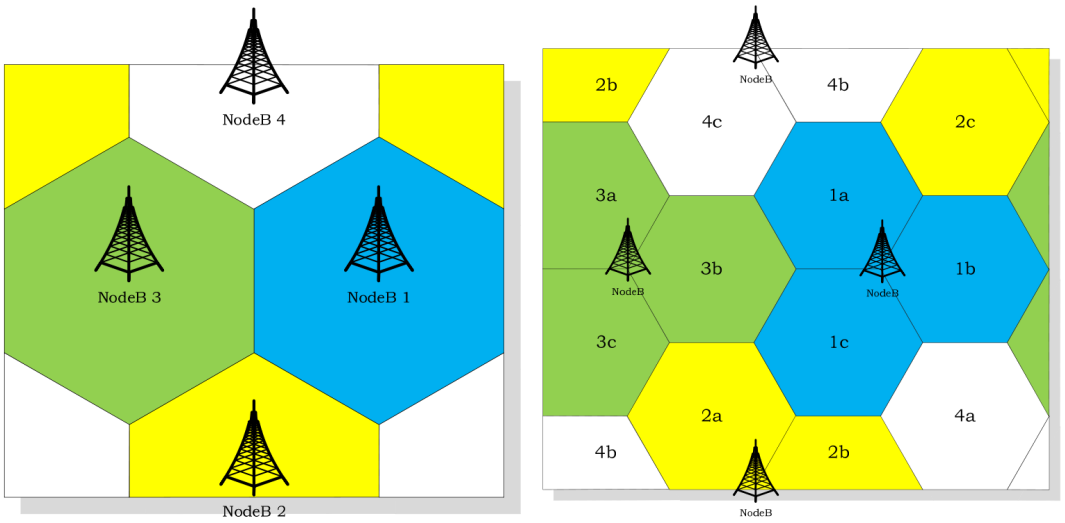
The traffic generation inside the NT is made by the Traffic Generator (TG) [10] that creates the network load. This traffic generation can be done following the different traffic classes that have been defined by 3GPP and which need to be supported in UMTS (see Section 2.6).

The traffic generator uses stochastic distributions of packets carrying dummy data to mimic at the best traffic patterns from real life applications. This way, there is no need to create many real life user sessions. However, it is still possible to introduce a real life application session

during an emulation. This allows the evaluation of the user's QoE. For example, introducing a real video streaming session while the emulation is running would give a good hint of what a UMTS user would experience under such circumstances.

3.3.4 Mobility Management

In the previous representation of the emulated world (see Fig. 3.1), NodeBs were considered as omnidirectional (Fig. 3.3(a)). In fact, they are composed of three sectors emitting in different directions (Fig 3.3(b)).



(a) Omnidirectional NodeBs.

(b) Trisectorial NodeBs

Figure 3.3: Emulated world representations. [1]

To avoid boundaries (if a UE "leaves" the emulated world), the testbed uses a wrap around technique. This way, "if a UE is about to leave the emulated world by crossing one edge, it reappears at the opposite edge (still within the same cell if necessary), as if moving on a torus (Fig. 3.4)." [1]

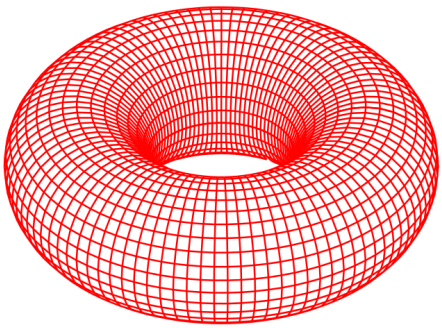


Figure 3.4: Torus world. [1]

Inside this emulated world, "UEs are able to move with four different fixed speeds defined by the 3GPP in [11]: 3 km/h representing a pedestrian, 30 and 70 km/h, which represents a UE on board of a vehicle within city limits and 120 km/h, which represents a UE on a train or on a car on the speedway." [1]

The UEs move following a random walk scheme based on the Gauss-Markov mobility model presented in [12, 13]. The UEs motions follow some real physical constraints. For example, a vehicle moving at 120 km/h cannot turn as abruptly as a pedestrian.

Of course, the NT manages handovers and interference. When a UE is moving, it has to stay connected to the network, and so to a NodeB. Handover is the mechanism which appears when a UE switches from a NodeB to another NodeB.

Chapter 4

State of the Art

This chapter aims at describing the different works and tools prior to our work. Tools and publications about the way to collect information on the access network have been explained; this is the base of our work on the environment. The second part of the Thesis is to perform several experiments. Because our experiments aimed at validating similar experiments described in [1], we have not reviewed the literature for references related to user experience in 3G networks.

Before starting to develop the approach of the NT validation, a choice has been made. Actually, there are different ways to work with a UMTS connection:

1. Direct access to a cell; it is necessary to obtain an agreement with a manufacturer or an operator.

2. Regular access to a commercial network by means of any Network Interface Card (NIC): PCMCIA card, USB card, mobile phone, etc.

3. Use of a Femtocell [14]. This is a new technology of indoor, residential cells. It is based on the same idea than a WiFi hotspot but on the 3G technology.

As the goal of this Thesis is to validate a UMTS emulation, we had to choose among these three possibilities.

The direct access is no longer an option: in the past the CINI laboratory had been granted access to an Alcatel test cell, but for now Alcatel has suspended the access right.

For the Femtocell, it is too early to purchase hardware as a residential customer.

Therefore our approach is focused on a commercial access. In addition, we will see that the CINI laboratory has performed different tests on UMTS (refer to Section 5.6).

As mobile phone is the most widespread hardware in relation with cells, tools have been developed to use this interface through the network. `cellTrack` has been identified as such tool [15].

As we see in Fig. 4.1, this software reveals useful information: cell ID, signal strength, etc. Nevertheless it has been only implemented for Symbian OS; it was therefore not possible for us to use it.

Operators are used to check the quality of their network using professional TEst Mobile System (TEMS) product. This kind of mobile phones integrates functions to detail the

environment. This is the case for the experiments performed in Vienna [16, 17] . Three different operators have been used at several locations. The method takes measurements on UMTS network especially about the transport channel. The necessity to use a TEMS product is not compatible with our approach.

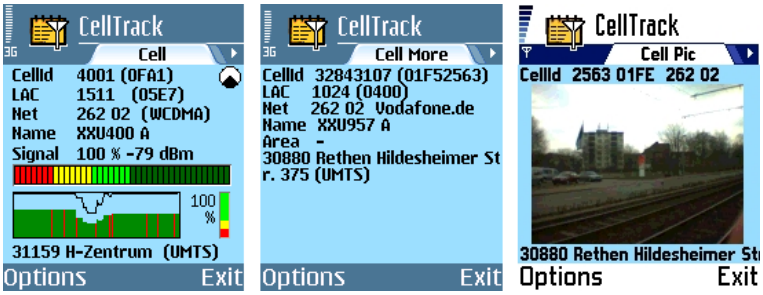


Figure 4.1: CellTrack on Symbian OS [15].

Except for the work over UMTS through a PlanetLab node (refer to Section 5.6), to the best of the authors' knowledge, no article in the literature jointly addressed environment tests over a 3G connection with a PCMCIA at the time we designed our experiments in Autumn 2008.

Since then, different articles have been published, describing similar experiments [18, 19]. In particular, [20] presents results of tests in 3G networks. The configuration was the same, the technology was UMTS and the main objective was to determine the transport channel. Their method was only focused on this parameter. It was too late to integrate their solution, nevertheless, our results indicate that this approach would not have changed much (refer to Section 7.5.1).

Therefore our approach is based first on the work realized by the CINI laboratory as later detailed in Chapter 6.

Chapter 5

PlanetLab

5.1 Introduction

This chapter aims at describing the PlanetLab (PL) environment. This environment has been used to perform a part of the experiments. We are going to describe its features and discuss its merits.

The CINI laboratory (where the tests have been performed) is registered in OneLab project where it is responsible of the deployment and the integration of the UMTS connection on a PlanetLab node; to realize the project they use an approach with Private OneLab. Finally we explain why PlanetLab has been integrated in our tests.

5.2 PlanetLab presentation

PlanetLab is an open, globally distributed platform for developing, deploying and accessing planetary-scale network services.

PlanetLab's goal is to support both short-term experiments and continuously-running network services, and ultimately to develop and demonstrate a new set of network services at planetary scale. It is available as a testbed for computer networking and distributed systems research. It is not a grid infrastructure, a similar or a research network separated from the internet [21].

The PlanetLab Consortium is established to support and enhance the PlanetLab platform in meeting these goals. It is responsible for overseeing the long-term growth of PlanetLab's hardware infrastructure; designing and evolving its software architecture; providing day-to-day operational support; and defining policies that govern its appropriate use [22]. PlanetLab members actively participate in developing tools for the greater good of the community, and as a result each user has a wide choice of tools to use in order to complete regular slice maintenance tasks.

This is a group of computers, using internet as a communication media, It was established in 2002. As of March 2009, PlanetLab was composed of 913 nodes at 460 sites worldwide (see Fig 5.1).

The architecture of PlanetLab is based on two important levels: the node-level and the network-level. The first one means that on each node several virtual machines can run at the same time, each one offering a different service. We have to note that the resources are distributed fairly and that the services are isolated from each other. Concerning the second level

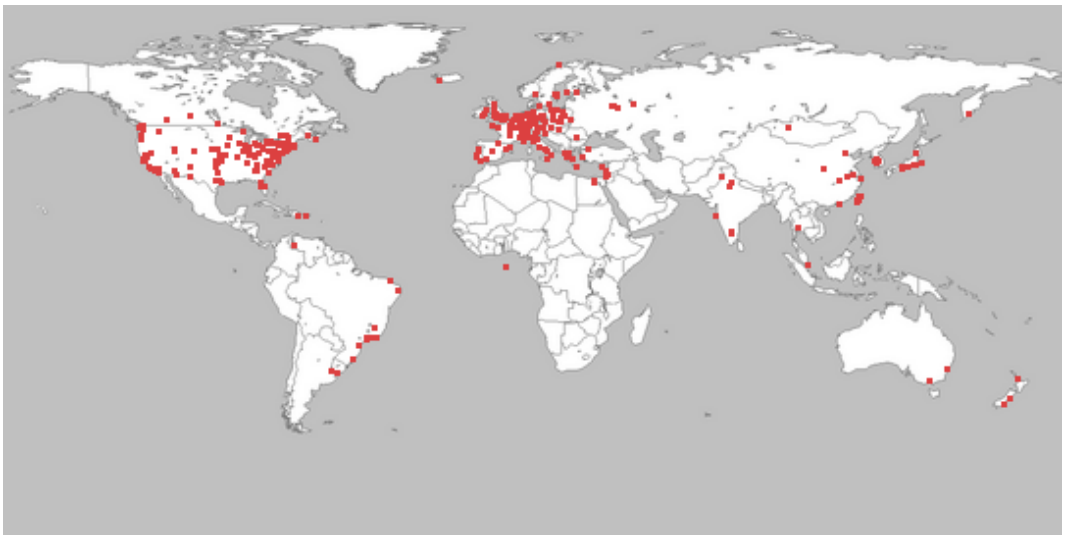


Figure 5.1: PlanetLab Nodes in the world (March 2009) [23].

(network-level), it manages the different nodes, it groups the authorities for the management and the slices [24].

In distributed testbeds, a portion of the testbed resources, called a slice, is assigned to each experiment. Slicing is usually implemented by means of virtualization techniques.

The slices are a set of virtual machines at different PlanetLab Nodes. An instance of a slice on a certain host is called a sliver. The authorities control softwares running on the node and check the security.

Regarding the slices, the authorities manage the namespaces on nodes and create the slices for the users [25].

5.3 Features of this testbed

First of all, the network size can be underlined. Indeed, the number of nodes available for the applications executions is impressive.

The next point is the node geographic distribution: PlanetLab enables to test new ideas under real Internet conditions which involve a large variety of non-controllable conditions.

Finally, the node conditions are also peculiar: machines shared by a large number of users and a node can die, can be updated, etc. We can mention a few other competing testbeds which have not these very features of PlanetLab: Orbit [26], Emulab [27], Vini [28].

The objectives followed by PlanetLab are to provide a robust environment to the variations of the resources availability. Therefore, the user obtains also a realistic execution environment which enables to replicate tests in controlled conditions. The main goal of PlanetLab is to produce a “real” network. PlanetLab can support seamless migration of an application from an early prototype, through multiple design iterations, to a popular service that continue to evolve.

This infrastructure is able to support different experiments at the same time. A node of PlanetLab works with shared resources as the CPU, memory, storage, etc. Obviously, there are several points which are required and described in Table 5.1. These requirements, specifications

are described by the management of PlanetLab composed of Princeton University, University of California and University of Washington.

Table 5.1: Minimum CPU, RAM, and storage requirements for a node [29].

	Until 12/31/2007	**1/1/2008 to 12/31/2008**	**1/1/2009 to 12/31/2009**
CPU	Intel Pentium 2.4Ghz AMD 2400+	Intel Pentium 3.2Ghz AMD 3200+, or 2.4Ghz MultiCore/DualCore	Intel Xeon 30x0 2.4Ghz (Mcore/Core2Duo)
RAM	1 GByte	4 GByte	4 GByte
DISK	180 GByte	320 GByte	500 GByte

The interface to manage a node, the users, the slices, etc. is a web interface. The following screenshot (Fig. 5.2) gives a thumbnail of the administration of a node.

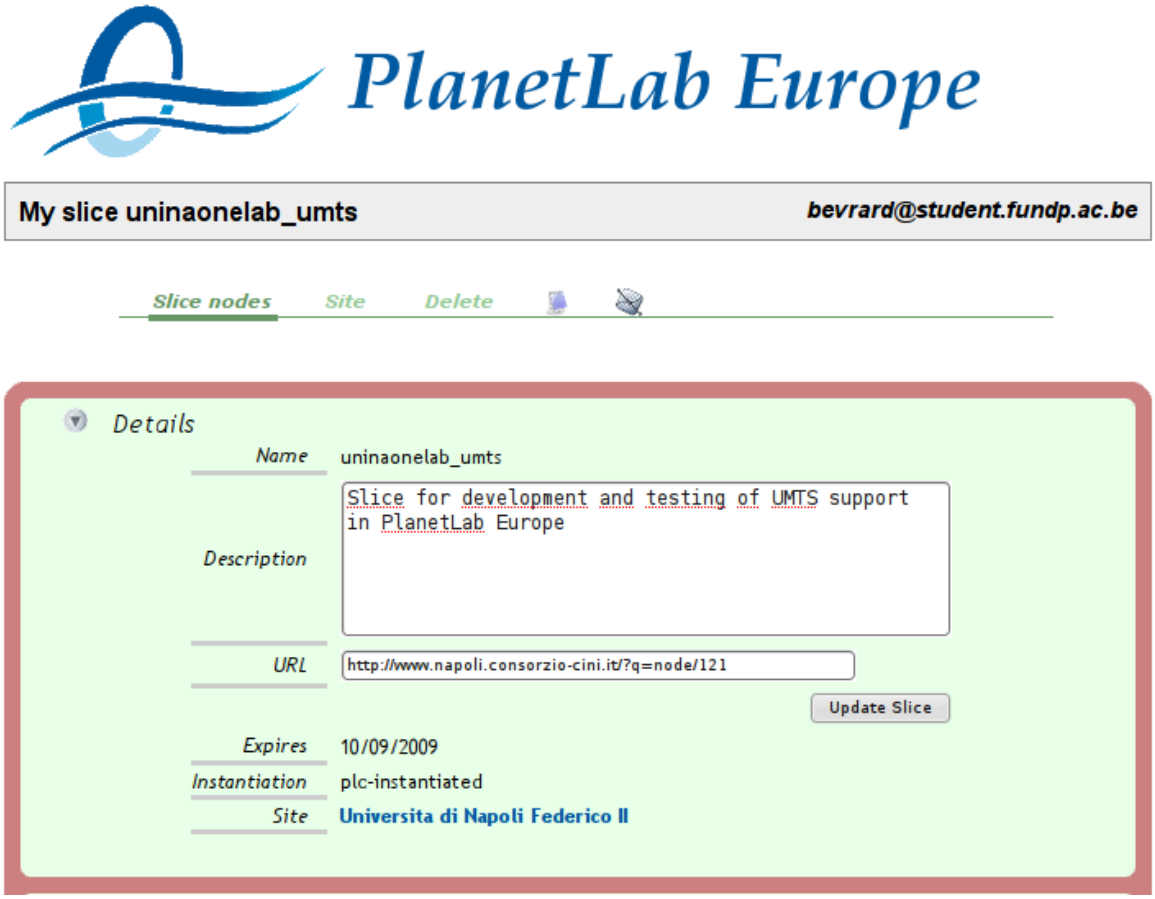


Figure 5.2: Screenshot of the web interface.

5.4 PlanetLab and OneLab

OneLab is a European Project, funded by the European Commission. The project started in September 2006 with two overarching objectives: to extend the current PlanetLab infrastructure and to create an autonomous PlanetLab Europe (PLE) (migrating European nodes and slices to an independent EU authority) [24].

PlanetLab Europe is a Europeanwide research testbed that is linked to the global PlanetLab through a peer-to-peer federation:

- PlanetLab is engaged in a federation trial with the OneLab project.
- OneLab is federating PlanetLab Europe which extends the PlanetLab service across Europe.
- OneLab is federating with other PlanetLab infrastructures worldwide (e.g. PlanetLab Japan) and with other types of testbeds.

The OneLab project supports network research and evaluate design solution for the future Internet technologies. OneLab is extending PlanetLab Europe into new environments, beyond the classic wired internet. OneLab is deepening PlanetLab Europe by incorporating new monitoring tools and new functionalities. It is a selected PlanetLab but it adds diversity [30]. The objectives of OneLab is not to start from scratch. It started from an existing testbed, namely PlanetLab (PL) (see Fig. 5.3). It also works on the "validation/integration" idea; to be sure that once the functionality is added, it can be deployed on every node. OneLab has the merit to have an international exposition, just like PlanetLab.

Extending PlanetLab, it adds wireless capabilities to the kernel: support for WiFi, UMTS (project of the CINI laboratory), WiMax, wireless ad hoc networks, emulated (for emerging wireless technologies). New features and technologies are integrated into the system when they become available by means of Private OneLab (see Section 5.5). It focuses on a PlanetLab everywhere, Internet Protocol (IP) on everything!

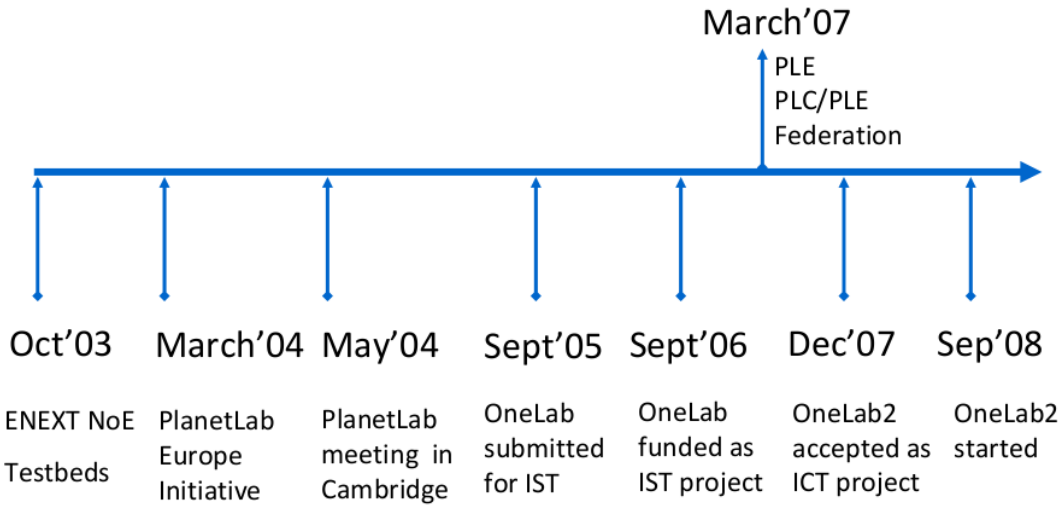


Figure 5.3: OneLab project's evolution [31].

5.5 Private OneLab and PlanetLab

The objective of extending PlanetLab has been pursued in OneLab through development of a number of extensions to the PlanetLab architecture, aimed at enriching it with new wireless access technologies, with powerful traffic monitoring instruments and with new emulation capabilities [32].

These new features have been implemented and validated in a project-wide distributed testbed, the so-called Private OneLab, whose resources are provided by and accessible to OneLab members only. Private OneLab serves as an incubator of new features to be eventually included in the public infrastructure. As these new features are thoroughly tested and validated, they can be later moved into the public PlanetLab Europe testbed [24].

One project of OneLab was the integration of the UMTS connection into a PlanetLab node. This objective is assigned to the CINI laboratory. For this new possibility, the team decided to choose a public connection because it is widely available and better suited to replicate real world scenarios. In this configuration, the Ethernet interface is used for control data and UMTS for the experiments. A slice user can use the UMTS connection [23].

5.6 Why integrate PlanetLab into our experiments?

The CINI laboratory has deployed its UMTS connection on a Private OneLab node which is based on a Linux system.

It is interesting to take an advantage of the UMTS link through the PlanetLab network. The objective is to add a new parameter in the validation: the hardware and the system (see Section 6.1) for the hardware differences. It is possible that both new elements involve the results? The cause could be the kernel which is different (on OneLab, it is a modified Linux kernel); an other one could be the architecture of the system which involves a limited root access on OneLab (problem met by the CINI to establish the UMTS connection), etc. The following tests and experiments take into account this parameter and evaluate it.

5.7 Summary of the projects and acronyms

Figure 5.4 and Table 5.2 illustrate the different links existing between the projects.

Table 5.2: Projects around PlanetLab [33].

Project Name	**Acronym**	**Description**
PlanetLab	PL	Project name of the international testbed
PlanetLab Central	PLC	Reference to the main PlanetLab, federated in the USA
OneLab	/	European research project improving PLC
PlanetLab Europe	PLE	European PLC administered by OneLab (UPMC)
Private OneLab	PrivateLab	Private experimental PlanetLab, new features for PLE

Finally we can mention Private PlanetLab. It groups each project which link to PlanetLab Central (PLC): AT&T, EverLab, KAIST, OneLab, Polish Telecom, University of Tokyo, Vini.

In the following chapters, PrivateLab is used to mention Private OneLab; PlanetLab means PlanetLab Europe. For the different schemas, these abbreviations are used: PlanetLab Central (PLC), PlanetLab Europe (PLE), PlanetLab Node (PLN) and PrivateLab Node (PRN).

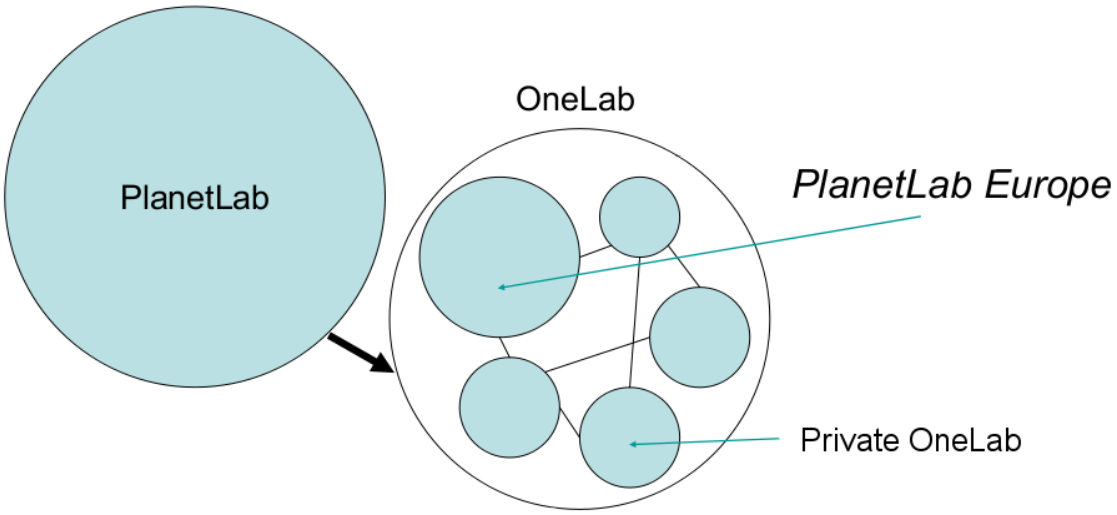


Figure 5.4: Links between the different projects [31].

Chapter 6

Description & Initialization

The main goal of this Thesis is to compare the UMTS capacities in a real environment with the situation of the NT. The tests are deployed on a PlanetLab environment, it is a static approach. Then, in addition, we are going to see if this PlanetLab configuration has an influence on the UMTS connection; to compare the results a common Linux-based laptop is used as a reference.

6.1 Global architecture

In this section, we explain the situation of the laboratory in relation with the UMTS connection and the PlanetLab Europe (PLE) association. This association is a group of computers available as a testbed (see Section 5.1). PrivateLab is a small PlanetLab like and is used at the CINI laboratory in order to test features before introducing them into the PlanetLab Node (PLN) (see Section 5.5 and [34]).

The main node to use a UMTS connection is the PrivateLab Node (PRN). As explained in Section A, we obtain a UMTS IP address through a `ssh` connection on the PrivateLab Node and few dedicated commands. In the case of the Linux Box (LB) we use our own script. After these steps, we have an Internet access via the UMTS Personal Computer Memory Card International Association (PCMCIA) card on PrivateLab.

6.2 Hardware

As shown in Fig. 6.1, we work on three systems: a LB, a PlanetLab Node and a PrivateLab Node. In addition, we use a Vodafone PCMCIA card for our UMTS connection, and for this one we have access to three Subscriber Identity Module (SIM) cards.

The Linux Box makes reference to a Toshiba laptop with Fedora Core 6 as operating system. A PCMCIA card is also installed with both interfaces: `/dev/noz0` and `/dev/noz2`. We have a full access to this laptop.

Regarding the PlanetLab Node, we work on the `onelab09.inria.fr` node on which we are logged in `uninaonelab_umts` slice.

Concerning PrivateLab, we work on the `onelab03.dis.unina.it` node. On this node we have access to the `unina_umts` slice and also the `root` account. A PCMCIA card is installed on `onelab03.dis.unina.it` and then we have access to the same interfaces that on LB: `/dev/noz0` and `/dev/noz2`. It is also important to take into account that the "OneLab" operating system is based on a Fedora 8 distribution.

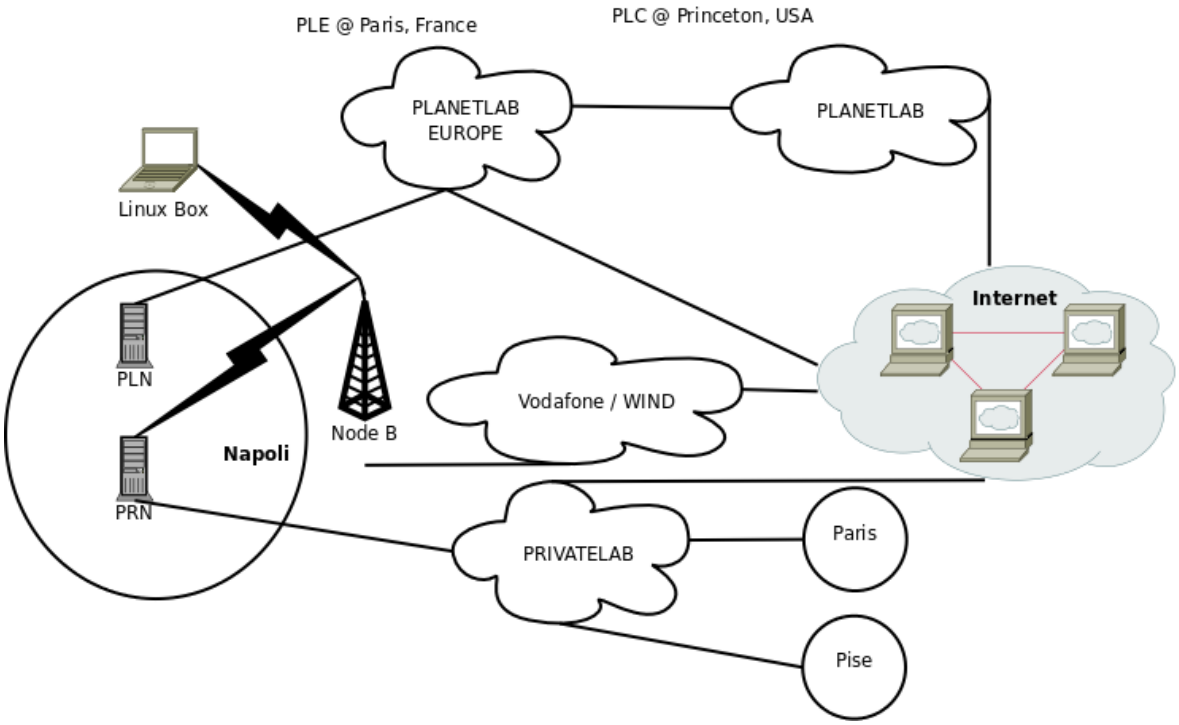


Figure 6.1: Global view.

In order to compare easily the different hardwares, Table 6.1 summarizes the situation.

Table 6.1: Hardwares of the systems.

	CPU	Cache	RAM	OS - GNU/Linux
Linux	Intel(R) Pentium 4(TM) 2 GHz	512 KB	512 MB	2.6.18-1.2798.fc6
PlanetLab	Intel(R) Xeon(TM) 2.80GHz	512 KB	3804 MB	2.6.22.19-vs2.3.0.34.29.onelab
PrivateLab	Intel(R) Celeron(TM) 366 MHz	128 KB	256 MB	2.6.22.19-vs2.3.0.34.28.onelab

The hardware is deployed in the laboratory as shown in Fig. 6.2. The main point to note, is that the Linux Box is at the center of the office and elevated to 1 meter above the ground. For the PrivateLab PCMCIA card, it is 20 cm above the ground and close to the wall.

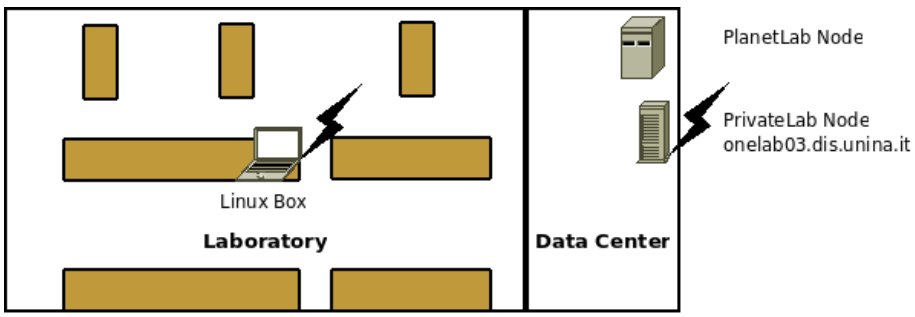


Figure 6.2: Laboratory at the University of Naples.

6.2.1 PCMCIA card

In order to obtain a UMTS connection on a PrivateLab Node and on the Linux Box, we need to host a UMTS NIC (see Fig. 6.3).

Manufacturer: Option Wireless Technology

Model: GT 3G+ EMEA

Hardware Revision: 2.2

Figure 6.3: Vodafone UMTS card.

6.2.2 SIM cards

In our environment we will work with three different SIM cards. These cards come from two distinct providers: Vodafone and WIND. We can also note we have three different contracts: Vodafone post-paid (VPOST), Vodafone pre-paid (VPRE) and WIND pre-paid (WIND).

In this Thesis we use both providers to illustrate explanations, tests and experiments without making a difference (the exception is the firewall problem which concerns only the Vodafone provider, see Section 7.4).

6.3 Setup of a UMTS connection

As we need a root access to setup the connection, different scripts have been written by the CINI Lab team to work on a PrivateLab Node. Indeed, the Chapter 5 explained that to deploy a new hardware on PLE, it takes a long time to be sure that there is no problem, it needs to be validated. This reason involves the usage of the PRN of the CINI laboratory which is based on a similar structure.

Nevertheless, the second node used for the tests and experiments is a PlanetLab Node because it does not need a special hardware (`onelab09.inria.fr`). Therefore, on the PrivateLab Node, the scripts allow a user working on a particular slice to obtain the root access (necessary to use UMTS connection) for a defined program [35]. IT is not the same for the Toshiba Laptop since we have a full access.

6.3.1 Nozomi Driver

First of all to use the UMTS NIC on both environments (LB and PRN), the PCMCIA card must be recognized as a device by the operating system. As described in [35], it is necessary to install the *Nozomi* driver which is freely available at [36]. We need to compile the driver and to add the module into the kernel directory. Once the driver is loaded and the UMTS NIC plugged in, four new devices are detected if the card is installed correctly.

Only two ports (#0 and #2) can be used:

```
[root@localhost ~]# ls /dev/noz*
/dev/noz0  /dev/noz1  /dev/noz2  /dev/noz3
```

The advantage of having two ports is that you can have your Point-to-Point Protocol (PPP) link on one port (refer to Section 6.3.4) and several AT commands on the other port. For example, you can monitor the data connection, check the signal strength or the registration without breaking the PPP data link.

6.3.2 The PPP protocol

Assuming the NIC is an area covered by the provider of its SIM card, PPP is used to set-up the UMTS connection. PPP is the protocol used for establishing Internet connections by using dial-up and Digital Subscriber Line (DSL) modems, as well as many other types of Point-to-Point Protocol links. The `pppd` daemon works together with the kernel PPP driver to establish and maintain a PPP link with another system (called the peer), and to negotiate IP addresses for each end of the link. The daemon can also authenticate the peer and/or supply authentication information to the peer.

6.3.3 wvdial

`wvdial` is a PPP dialer, which starts the PPP protocol in order to connect to the Internet, interacting with the local PPP daemon. Once installed, it is necessary to edit the configuration file `/etc/wvidial.conf`. In fact when `wvdial` starts, it first loads its configuration from this file which contains basic information about the modem port and init string, as well as information regarding the Internet Service Provider (ISP), such as the phone number, the user; name, and the password.

```
#/etc/wvdial.conf

[Dialer Defaults]
Phone = *99***1#
Username = any
Password = any
Stupid Mode = 1
Dial Command = ATD

[Dialer pcmcia]
Modem = /dev/noz0
Baud = 460800
Init1 = ATZ
Init2 = ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
ISDN = 0
Modem Type = Analog Modem

[Dialer 2gonly]
Init3 = AT+COPS=0,0,"I WIND",0

[Dialer 3gonly]
Init3 = AT+COPS=0,0,"I WIND",2

[Dialer wind]
Init4 = AT+CGDCONT=1,"IP","internet.wind"
```

Details:
The `*99***1#` is the provider's phone number. The user name and the password are not defined because they are hard coded in the SIM card.
`Stupid Mode` set on 1 tells `wvdial` not to wait for a prompt, just start `pppd` immediately; this is required because there is no authentication.
Next we find the `Dial Command`. On the web, we can find two commands `ATDT` and `ATD`. We chose `ATD` because this is the standard given in [37]. For the configuration of the modem, we ask to `wvdial` to work with `/dev/noz0` interface of the UMTS NIC.
`Baud` is the speed at which `wvdial` will communicate with the modem.
`wvdial` can use up to nine initialization strings to set up your modem. Before dialling, these strings are sent to the modem in numerical order.

`Init 1` is `ATZ`, it initializes the modem. Then `Init 2` groups different commands which configure the echo, verbose, character modes, also the way to answer. `FCLASS` informs that some fax or voice capabilities are present.

For the third step we can make a choice for the configuration of the provider. The first `0` specifies that we configure the home network; the second one the way to define the provider (short name, long name, etc.); and then the technology that we use (`0` is for GSM and General Packet Radio Service (GPRS) and `2` is for UTRAN). The exact name of the provider is obtained by the AT command `AT+COPS=?` (as explained in Section 6.3.4).

The last step of the configuration defines the Packet Data Procotol (PDP) context where we have to specify our operator's Access Point Name (APN) (in this case, it is the APN of the WIND provider). The information about the AT commands can be found in [37] and for the `wvdial` configuration [38].

6.3.4 gcom and minicom

Before starting a PPP connection, the UMTS NIC must be initialized and registered on the network. For this purpose we can use `gcom` or `minicom` to check the registration.

`gcom` is a scripting language interpreter, useful for establishing communications on serial lines and through PCMCIA modems as well as GPRS and 3G datacards.

`minicom` is a lower level program, a terminal which can directly dial with the modem. It has the great advantage to interpret AT commands without any script.

Once the UMTS NIC is plugged in, the Light-Emitting Diodes (LEDs) (blue and red, for the Vodafone Option card) of the card will be turned on; the SIM card is registering to the network. By default the card prefers a UMTS network but if it does not find a UMTS network (blue LED), the card will choose a GSM network (red LED). If we want to have more information about the registration, we can list all available providers (with the `operator` script of `gcom` or with the AT command `AT+COPS=?` in `minicom`) and then look at the network that the card has chosen:

```
# minicom #
at+cops=?
+COPS: (2,"I WIND","I WIND","22288",2),(1,"I WIND","I WIND","22288",0),
(1,"vodafone IT","voda IT","22210",0),(1,"vodafone IT","voda IT","22210",2),
(1,"I TIM","TIM","22201",0)

OK
```

We can see that in our case few networks are available. We can look in details the information of our provider:

`(2,"I WIND","I WIND","22288",2)`

`2:` current network. 1 is for an available network.
`I WIND:` is the short name of the provider.
`I WIND:` the long name (the same in this case).
`22288:` is the provider ID code.
`2:` is the technology used. Here, it is UTRAN; GSM is 0.

Then we can check the registration on the network (with `gcom` or the AT command `AT+COPS?`).

```
# minicom #
at+cops?
+COPS: 0,0,"I WIND",2
```

`0:` means that we are in the automatic mode.
`0:` we use the long name of the provider.
`I WIND:` the provider.
`2:` the techonology used (UTRAN).

The last information is very important because if the registration is made on the GSM mode, after the `ppp` initialization, we will get an IP address but on GPRS and not UMTS.

The solution to avoid this problem is to put the card only in UMTS mode. Hence GSM networks will not appear. We can always match `gcom` and the AT command in `minicom` (`3G` is the script, and it uses `AT_OPSYS=1`). When completed, it will be the default setting.

6.3.5 UMTS connection

In the following paragraphs we describe the process to obtain the connection on both different hardwares with a UMTS cell (see Fig. 6.4).

Figure 6.4: GSM, UMTS cells.

Linux box

Now that each step has been described, we can have a look on the commands used on the LB. First special part in the connection is the initialization. It initializes with a script (see Appendix A), when the card is plugged in.

```
root@...:sh umtsInit.sh
```

This script prepare the connection: it loads the module in the kernel, it tests the interfaces and also checks the available networks.

The second element is the command `route add IP`. Next to the connection, this command force the use of the UMTS (following the IP address).

PrivateLab node

For this hardware, we will connect us with the scripts which are designed to make easier the connection process. As described in the paper of the CINI Laboratory [34], the script to establish to connection will use the following commands:

`bash: umts status`	This command checks the status of the connection.
`bash: umts start`	We attempt to connect the card on the network. If we get an IP address, it works.
`bash: umts stop`	Command to stop the UMTS connection.
`bash: umts add IP`	This command is similar to `bash: route add IP ppp0`. By default the system use the `eth0` interface so we need to specify which IP address must go through the `ppp0` interface (see A.2, end of the script).

Connection

The connection is realized by a common script for both systems: run `umtsStart.sh` (see Appendix A.2). This script contains commands or scripts previously defined. Depending on the inputs (provider and directory which defined the hardware), the script initiates the connection with `umtsInit.sh` for LB and `umts start` for PRN. Next it continues to make a difference between both systems with commands like `route add IP` and `umts add IP` to define a route used by the UMTS.

This script produces an output which contains an IP address: the connection is established (appendix A.3). We can note that the same script is used on both systems; the ways are different but the script chooses following the system.

6.3.6 Issues

Two interfaces

The use of the AT commands is helpful to obtain information on the network. As we know, we can use it to scan the networks, configure the card, etc. but one problem is the impossibility to use them when the connection is established. Imagine that in a terminal we run `root@...:wvdial umts wind` or that the card is configured in `minicom` and we use `ATD*99***1#`. In both cases, the system will answer `CONNECT` and then it will be impossible to type any AT command to collect any information during the connection.

In fact, this is normal, because when `ATD` is used, the card waits for a PPP connection, for an answer to `pppd`. It is locked on the daemon, but it is the `/dev/noz0` which is busy. As explained in [36], we can use the other interface to monitor the data connection, check the registration, etc. without breaking the PPP data link.

What is the real network used?

In general, each hardware has an automatic mode in its configuration. This is also the case for the UMTS NIC used at the CINI laboratory. So, it is possible that when the card is plugged and the registration checked, the obtained answer is : `+COPS: 0,0,"I WIND",0`. It means that we are on the WIND network, in the automatic registration and in the GSM mode.

After this registration, the second step is to use `wvdial` to obtain an IP address. The answer is positive but is it a UMTS network? The documentation [37] shows that it is the GSM/GPRS mode; to be on the UMTS/UTRAN, we need to be registered on: `+COPS: 0,0,"I WIND",2`.

If this network is available, what is the problem?

As explained in [37], we tried to force the network like this: `AT+COPS=1,0,"I WIND",2` which uses the manual mode for the registration, but it did not work.

The specification of the AT commands [37] and the official commands of Option specifications [39] did not give information any more on the choice of the technology. Finally we found in [36] and in an other page of the Option support that there was a proprietary command (non defined in the specifications) which could force the technology: `AT_OPSYS=1` (then the card works only on the UTRAN mode). This option was also specified in `gcom` behind the `3G` script.

In that case it is important to note that an automatic configuration is not sufficient and it needs the special AT command to reach the UTRAN network.

Once the card is locked on UMTS, the failure to get registered is only due to the weakness of the radio signal.

Chapter 7

Environment - Tests

In this chapter we work out different tests in relation with the environment defined in the previous chapter. If we want to evaluate the multimedia experiments, we first have to know the experimental network as best as possible. This is why different properties are selected and tested; they correspond to the UMTS characteristics of the NT.

One main issue is touched on this approach: the Vodafone firewall problem. It is essential to explain it because it impacts the capacities of the Vodafone network.

Finally, this chapter closes with an analysis describing the environment following the selected characteristics.

7.1 Properties of the network

First of all, we have to note that the UMTS connection is a commercial access (Vodafone and WIND - refer to Section 6.2.2). Therefore, we cannot control the network parameters. Thus, we have to know the possibilities and the properties of this network in order to decide what kind of scenarios we can compare with the NT.

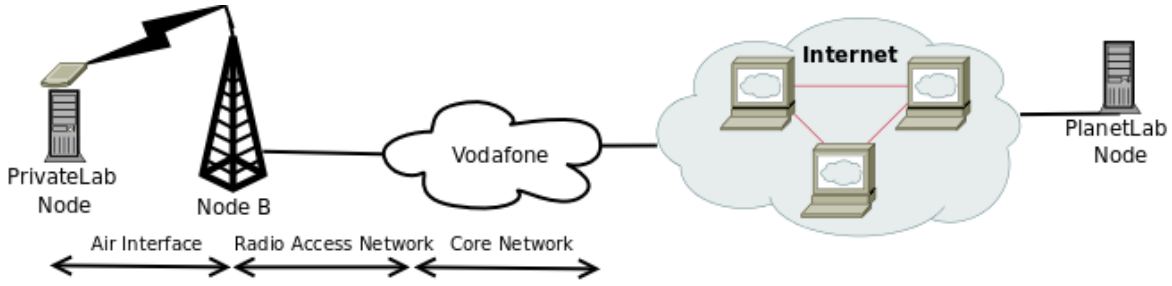


Figure 7.1: Configuration for the environment tests.

In this configuration we have to know the following parameters. For each parameter, we will describe why we need it (if useful for the environment, defined on the NT, etc.) and also how we will collect its setting. These parameters are selected to compare the real situation to the NT as much as possible. This approach is realized on the configuration shown on Fig. 7.1, where the PCMCIA card is plugged in the PRN. Vodafone is used as reference in figures, nevertheless it can always be replaced by WIND [1].

[1] For the tests on the Linux Box, the PRN is replaced by the laptop

Spreading Factor: The SF defines how many chips are used to code one user data symbol (refer to Chapter 2).
Why? This information will regulate the bandwidth because higher SFs will subsequently reduce the data rate and lower SFs will increase the data rate.
How? In order to evaluate the SF, different measurements are taken on the bandwidth. Otherwise, the only possibility to know the value of the SF on a commercial connection (with a PCMCIA card) is to ask it to the provider or to derive it in an analytical way.

Transport channel: In the UTRAN, data generated at higher layers are carried over the air interface using transport channels. There are two types of transport channel: dedicated and common/shared channels.
Why? Here we work on the quality of the connection because a common channel is a resource divided between users in a cell, whereas a DCH is a reserved resource for a single user.
How? Like the SF, we can not discover the information. If the provider does not give the information, we are forced to work on assumptions.

Maximal downlink traffic: This is the limit of the bandwidth speed on download.
Why? Every network has its theoretical limit, we will check if we have a good coverage.
How? We will discover it with two different ways. The first one is to download a file of 10 MB. This way, the connection stabilizes itself and we can register the speed. It can be a good indication but it depends also on the web server (similar to Fig. 7.1 where the web server replaces the PlanetLab Node). The second way is the generation of synthetic traffic from a PlanetLab Node to the PrivateLab Node (see Fig.7.1).

HSDPA: The NT complying with UMTS Rel'99 has been realized on the UMTS standards. In our case, the Vodafone's network uses maybe the HSDPA standards.
Why? The great advantage of this one is to have a higher downlink speed but in our case, we will check which technology the providers have deployed.
How? For this property we will use the AT commands.

Strength of the signal: This information is determined by the PCMCIA card.
Why? This will be a good information on the quality of the connection. It can also help us to check the distance between the Mobile Terminal (MT) and the NodeB.
How? To discover these properties we can use three softwares: `UMTSmon`, `gcom` or `minicom`.

Position of the cell: Each NodeB has a GPS position or a direct address in the city.
Why? With this indication, we will obtain an estimated distance between the cell and the MT. We can also check how the signal strength is reduced because the signal is attenuated in relation to the distance.
How? We have two scripts that can show cells' position in `Google Earth`. These files are linked to a data base and we will check if our cells are listed. We have also access to the official list of all Vodafone's cells in Italy.

Load of network: As we will use a commercial connection, we can not control the network (number of users connected and kind of connections).
Why? In every network, each user takes a part of available resources. If there are many users at the same time, the quality of the network will be degraded. The load of network E2E is also important for the quality of the connection.
How? Whether Vodafone can help us and give an idea of the load of network; or we have to assume that the load of network is from medium to high at noon and low at midnight.

RAN Round-Trip Time: In the NT, there is no radio transmission and normally the Round-Trip Time (RTT) is much more important in a real UTRAN. To be close to the reality, it has been fixed to 50.2 ms [8].
Why? It can be interesting if we check this property because, like others, it simulates a real behaviour.
How? We can use `traceroute` program to try to check the time but we foresee that few machines will answer.

Downlink oriented: We have to keep in mind that the NT is downlink oriented; so in a TCP connection acknowledgements (ACK) are not disturbed, which is different from the reality.
Why? It seems normal that we will have a loss of quality in the real network, we would like to see if this characteristic is a low or high assumption.
How? During a generation of traffic, we will analyze the traffic of ACK packets. The loss of packets will determine the load of the assumption.

Types of users: In the NT, the SF allocation scheme is based on two main properties: the traffic class and the user profile (as described in [1, p.73]).
Why? It would be interesting to discover which kind of user we are in this real network because the quality of the link depends on this characteristic.
How? The AT commands can be useful. We have also to look at the contract of our SIM cards.

Radio Link Control Mode: the UMTS Radio Link Control (RLC) layer is able to transmit in three different modes : Acknowledged Mode (AM), Unacknowledged Mode (UM) and Transparent Mode (TM). The NT is focused only on two of the RLC modes: TM and AM (for details see Section 3.3.1). In our environment, we can not choose the RLC mode.
Why? Like other properties, this parameter has consequences on the quality of the connection.
How? Normally, with the AT commands, we can discover the mode that we use.

Traffic classes: The NT is designed for four traffic classes: conversational, streaming, interactive and background.
Why? It is possible that the provider does not allow a terminal to use all these classes. We have to check if we are not always in the background class.
How? It mights depends on the environment. In our situation, it is impossible to obtain this information with this hardware.

7.2 Tools

Here we are going to list all softwares or scripts needed to perform our tests on the PRN and on the LB.

7.2.1 Traffic Generator and Wget

For the generation of traffic, two tools are used. The first one is `wget` [40], which allows to download a file as fast as possible without going to the bandwidth limit; this is a "real" traffic, not forced.

The second tool is Traffic Generator (TG), `TG` [10] is a traffic generator program that creates one-way UDP or TCP streams between a source and a sink.

The traffic is described in terms of interarrival times and packet lengths. Information regarding the source and sink, such as packet transmit and receive times, is recorded in a binary log file for later post processing by `dcat`. `dcat` takes the binary log file and produces an ASCII representation. We can find in Appendix D.1 a simple example of a TG configuration.

TG is a generator that we can configure as we like. In that case, the traffic tries to reach the limit of the bandwidth; this is a synthetic traffic that we force to the limit.

The interest of `wget` is also that it uses the TCP protocol. This protocol cannot be used with TG (please refer to Section 7.4); `wget` is therefore relevant to compare the behaviour of TCP with User Datagram Protocol (UDP).

7.2.2 Tcpdump and Wireshark

`tcpdump` [41] prints out a description of the contents of packets on a network interface. It can also be run with the `-w` flag, which causes it to save the packet data to a file for later analysis. `tcpdump` has no graphical interface and it is suitable to the PlanetLab and PrivateLab environment. It is mainly used to capture packets but it is not appropriate to analyse packets. This is why we will also use `Wireshark` [42] which is a free packet sniffer computer application. It has the advantage to have a graphical interface and can easily provide statistics and graphics.

7.2.3 Scripts to establish the connection and gcom

In order to use the UMTS connection, we will connect us with the scripts which are designed to make easier the connection process (refer to appendix A for the scripts and Section 6.3 for commands detail). We can note that, behind the commands on PRN (`umts start`, `umts status`, ...), we can find the `gcom` software which allow us to initialize the PIN for the UMTS SIM card.

7.2.4 UMTSmon

`UMTSmon` [43] is a tool to control and monitor a wireless mobile network card (GPRS, EDGE, WCDMA, UMTS, HSDPA) in a laptop running the Linux operating system. It handles PIN codes, operator choice (roaming), signal strength and network statistics, sending/receiving SMS. This program is similar to `gcom`. Unfortunately it needs a graphical interface which is not available on a PrivateLab Node. In order to avoid problems between scripts developed by the CINI Laboratory and `UMTSmon`, we will focus on `gcom`. `gcom` has been introduced in section 6.3.4 along with `minicom`.

7.2.5 Google Earth network link and Vodafone database

If we want to know as best as possible the environment we need to know where the cells are. After many searches on the Internet we found these network links that we can use in `Google Earth` [44, 45]. This link estimates the location of GSM cells based on measurements that relate the cell ID of the current cell with geo-coordinates recorded with a GPS device. This location information can be used to provide people with an estimate of their location based on cell ID only.

In Section 7.3.3, we will see if the link and the cells displayed (like on Fig 7.2) match with our cell ID. For the Vodafone network, we have also access to the complete database of cells available in Italy. In these files we have the exact addresses of the cells.

Figure 7.2: Cells around the laboratory [45].

7.3 Tests

If the main goal of this Thesis is to validate the NT, we also have to check if there is a difference between a PRN and the reference - LB. For this purpose, we have to keep in mind that both machines are not in the same conditions (see Fig. 6.2). Nevertheless, both are static.

Here, we are going to check if the environment changes according to both systems. The schema of our tests will be detailed with the PrivateLab Node but it is the same procedure for the Linux Box.

The last point to note: each test will also be executed on the three different SIM cards: Vodafone post-paid (VPOST), Vodafone pre-paid (VPRE) and WIND pre-paid (WIND). The wired interface (ETH) will be used as a reference, and tested at 18.00. The tests on the UMTS connection will start at 12.00. In this situation, we assume the load of network as medium to high. It was also planned to perform the experiments at 4.00 (assumed as a low load of network) but due to limited time and limited SIM cards credits, we had to restrict our experiments on the 12.00 time slot. For the same reason, tests were run only once.

Each test was monitored by a main script in which it is possible to modify parameters like the system (Linux Box or PrivateLab node), the interface (`eth0` - wired connection, or `ppp0` for the UMTS connection), the provider, the date and the PlanetLab node (node used as destination). These parameters are necessary for the initialisation of the tests but also to save data automatically. For the detail of the script refer to Appendix E.1. It is important to note that, once run, this script starts each test and each experiment automatically one by one.

Now we will take a look at the different tests realized on the systems. We will begin with simple environment tests and then we will work on information requiring a scenario.

7.3.1 Test 1 - HSDPA

The most important information to determine at the top of the tests is: "is the UMTS or HSDPA technology deployed on the environment?". The difference has already been explained in Chapter 2; HSDPA is an evolution, the potential is well more important.

The more simple way to check this information is to find an AT command returning whether the connection is HSDPA or not. This kind of command is not defined in the 3GPP specification [37]. Nevertheless, the PCMCIA card manufacturer (Option) has defined different proprietary AT commands. One of them (`AT_OHCIP`, for the script see B.1) was available on their web site [39] and answer to: `""HSDPA Call In Progress"?"`.

Traffic must be generated to try this AT command because a "call in progress" is required.

Results:

LB : All results return HSDPA = 1 (used).

PRN : On the node it returns 0. Nevertheless, the other tests show characteristics of the HSDPA; no explanation has been found.

It is the first important result: every provider in this environment uses the HSDPA technology. The different tests which work out on the NT with UMTS parameters cannot be directly compared.

7.3.2 Test 2 - Strength of the signal

For this characteristic `gcom` or `minicom` can be used but we will focus our test on `gcom`.

These two programs use ETSI AT commands [37]. There are many AT commands standardized. One of them, signal quality `+CSQ`, returns Received Signal Strength Indication (RSSI) and channel bit error rate `<ber>` from the MT. This command is already included in `gcom`, we just have to put the parameter in the command line of `gcom`: `bash:gcom sig -d` *`device`*.

It is also possible to use AT commands to obtain other information about the location. With `gcom`, we can write short scripts which contain AT commands (by this way, see B.2.1 - we can do the same with `minicom`).

Table 7.1 summarizes the results.

Table 7.1: Signal strengths following systems and providers.

	`Signal strength - [RSSI]`		
System	VPOST	VPRE	WIND
PrivateLab Node	18.99	19.99	23.99
Linux Box	22.99	23.99	26.99

The signal strength returned a result in the RSSI form, indicates that the signal for the PRN is lower (average: signal = 20.66) than for LB (average: signal = 24.65). WIND has the best result, with a RSSI of 26.99 on LB.

7.3.3 Test 3 - Cell id

In this way we can try to use the `+CREG` and `+CGREG` commands [37, p.47 and 157] to obtain the location area code, the cell ID and the access technology of the registered network (UTRAN,UTRAN w/HSDPA,UTRAN w/HSUPA). The scripts of these files are defined in B.3.

Table 7.2 gives the results.

Table 7.2: Cell ID following systems and providers.

	`cell id` - [hexadecimal form]		
System	VPOST	VPRE	WIND
PrivateLab Node	026F	026F	0CB8
Linux Box	026F	026F	0CB8

The cell IDs are in hexadecimal format, we have to modify these values to obtain the cell IDs in the decimal form in order to match values with those in the database. We obtain 623 for 026F and 3256 for 0CB8. For now these values do not match with our data (see Section 7.2.5) but in the following sections we will see whether it is possible to match with a cell, once all properties are collected. The AT command should normally define the technology but it is not the case; it returns only the location area and the cell ID.

7.3.4 Test 4 - Traceroute

For this test we would like to know if it is possible to measure the RAN RTT and obtain time statistics on the data path. For security reasons, servers are likely to not answer.

In the NT the delay to the RNC is fixed to 50.2ms. So we are going to try to use `traceroute` over a Public Land Mobile Network (PLMN) (see Fig. 2.2). Next, we launch a `ping` on the first IP address returned by `traceroute` (for the scripts, see appendix E.3).

Results:

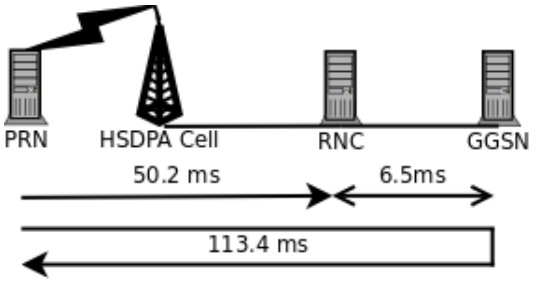


Figure 7.3: Delay through UTRAN.

The NT follows the 3GPP specifications but is it the reality? First of all, the 50.2 ms is the time to reach the Radio Network Controller (RNC) and in this case, is it the RNC which answers? Unfortunately, it is not this element of the network, this is the Gateway GPRS support node (GGSN) - the gateway to the Internet. Nevertheless, if we take a look to Fig. 7.3, the result (ping: 113.4 ms) is not too bad. Indeed, if we consider that the network follows the 3GPP specification, it only takes few *ms* to reach the GGSN since the RNC. In this condition, we can accept the assumption of the NT with reference to the results in the real environment.

7.3.5 Test 5 - RLC Mode

In this section, we try to discover the RLC mode. We have to find a way to know in which mode we are: synchronous, asynchronous or transparent mode.

This test is, actually, not possible following our conditions, our environment. The only approach to determine the mode (ACK or Transparent) would be this one: compare the traces of WIND and Vodafone for the same transmission, hope that both are disturbed, and see if there is a difference of time, sign of retransmission for one and not for the other.

This is unrealisable but we can note that the use of the 3G technology is not spread widely yet. Then we can speculate that WIND and Vodafone have chosen the easiest solution, i.e. Transparent, as long as it is not necessary to activate the ACK mode.

7.3.6 Test 6 - Type of the user

As explained in [37, p.68], it seems that it is possible to know if the SIM card has a priority level in the network. It will be impossible to change this level but we hope to know our level. On [37, p.69] levels begin from 0 to 4; it corresponds maybe to the four users profiles established in [1, p.73] with a "root" in addition. It is maybe another property, we have to check this characteristic.

We have to test these specific commands (for the script see B.4):

1. `+CAEMLPP`
2. `+CPPS`
3. `+CFCS`

Results:

This answer is the same for all three commands: an ERROR result code. This means that the user has no priority level, or that the command has not been implemented for this hardware. The 3GPP specifications ([37]) have defined the commands allowing access to the information, the data into the network. The problem here is that the hardware used is a "basic" hardware, not like a TEst Mobile System (TEMS) [46]. Therefore, there are not a lot of functions which are implemented.

In our case, user functions have not been selected for this hardware; this explains the ERROR result code.

In order to give a priority following the SIM cards used, the contract of the cards can be considered: two providers (Vodafone and WIND) and two different contracts (Vodafone Pre-paid, Vodafone Post-paid).

Other AT commands regarding the QoS are defined in the specifications ([37]). Nevertheless, like these about the type of user, they are not available for this kind of hardware.

7.3.7 Test 7 - Maximal downlink traffic

The main goal of this test is to make certain if we are on a UMTS or HSDPA network. This test also shows the limit of the network, the differences between the theoretical speed of the network and the reality.

For the generation of traffic, two techniques have been considered, but one of them has proved to be applicable. In the first time, `wget` is the generator. Next it is TG which is used. The difference and advantages between both tools are explained in 7.2.1.

`wget`

The configuration for this test is shown in Fig. 7.4 . We start a `tcpdump` session and then we download a file with `wget`. The script can be found in the appendix E.4.2. We can note that for the UMTS connection we have to modify the route to use the `ppp0` interface. Once we will have the `file.dump` (i.e. the trace) we will analyse it in `Wireshark` to obtain statistics. This test has been realized with a 10 MB PDF file (we use the "-o" option to obtain the figures, file to download `http://www.umtsstreamingexperience.be/validation/test.pdf`).

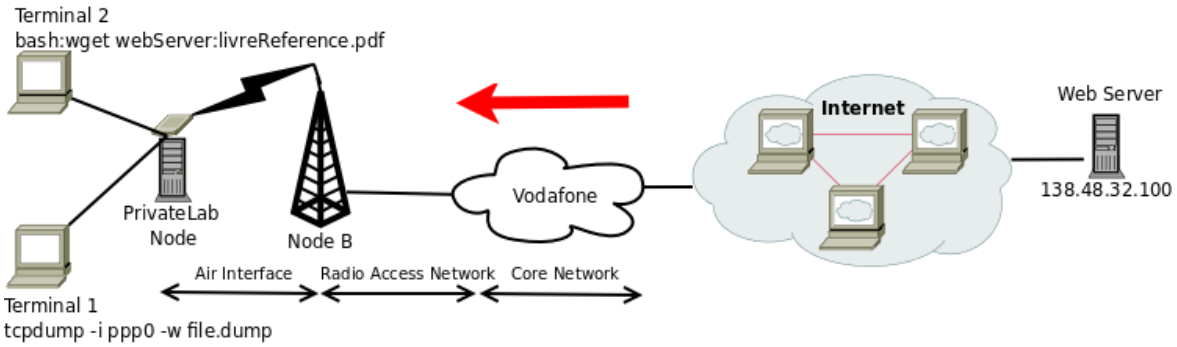


Figure 7.4: Environment for the test 1 with `wget`.

Results:

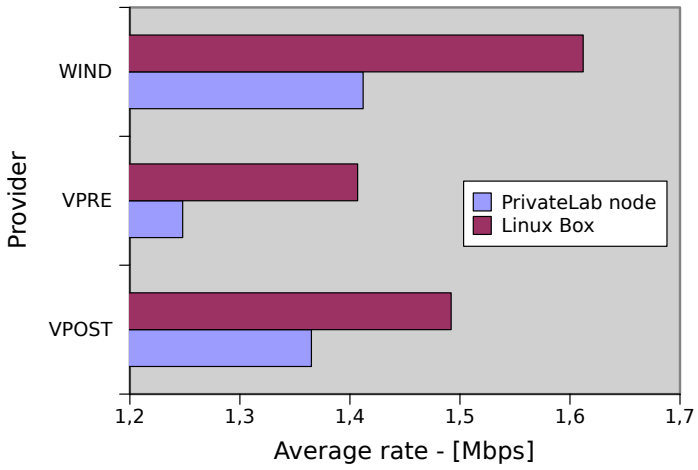


Figure 7.5: `wget` results.

As we have already seen before, the theoretical speed for the UMTS technology is 2 Mbps. Even if it can reach 7.2 Mbps or 14 Mbps following Fig 2.7 on the release 5, the environment is locked on 2 Mbps because, for now, it satisfies the users.

The Ethernet connection (ETH) is faster than the HSDPA connections: the results in appendix G.1.2 indicates that ETH is ten times faster on PRN and forty times faster on LB than HSDPA (this is why ETH does not appear in Fig. 7.5). This figure shows that we reach similar speeds in general, with an advantage for WIND (1.612 Mbps - see G.1.2). `wget` shows that for a real - no synthetic - transfer, it uses correctly (more than 75% of 2 Mbps allowed) the line without reaching the limit of HSDPA. The test confirms again the HSDPA environment and not a UMTS environment allowing only 384 kb/s for the downlink.

The Linux Box has an advantage on the PrivateLab Node, but it is not significative.

`tg`

This test uses a similar configuration on the PRN, TG-Server and `tcpdump`, and a TG-Client on the PLN - as we see in Fig. 7.6).

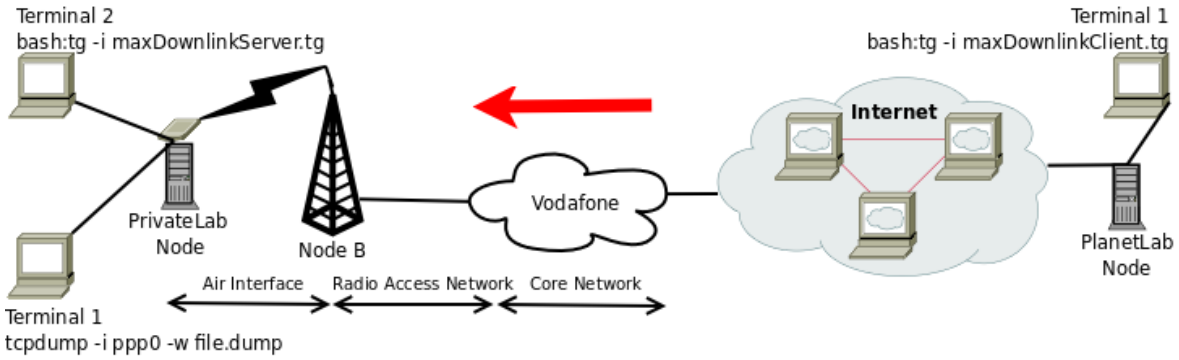


Figure 7.6: Environment for the Maximal Downlink traffic test with TG.

We can find the `tg` configurations and the main script in appendix E.4.1. There are two kinds of configuration. The first one tries to reach a speed of 2,000 kb/s (because the first test with `wget` suggests a HSDPA environment) with a packet size of 1,450 Bytes and the other one with a packet size of 250. For these tests the quality is not important then we can use the UDP protocol.

Results:

Here, the objective is to reach the limit of the connection. The Ethernet test indicates that the configuration of the TG files enables to reach an average speed higher than 2 Mbps.

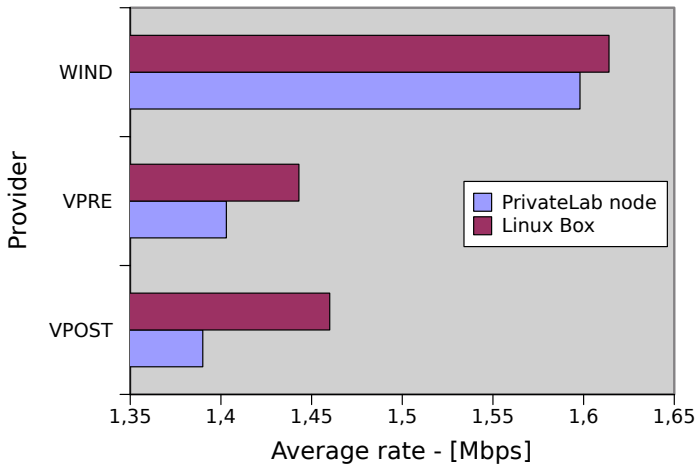


Figure 7.7: `tg` (2 Mbps, packet size: 1,450 Bytes) results.

The results are generally similar to the `wget` results, with again an advantage for the provider WIND (see Fig. 7.7 in the case of TG configured for 2 Mbps with a packet size of 1,450 Bytes). Another thing to note: with the use of small packets, the speed increases again by few kb.

About the packet loss, following the percentages (see G.1.2 - WIND with better results: 19,7% with packets for 1,450 Bytes and 31,6% for packets of 250 Bytes), it seems clear that if we use smaller packets for a higher speed, the packet loss rate will also be higher. In this case,

the objective is only to reach a high average speed; but in a real context this approach is not the best. It is important to evaluate the best average speed with a low packet loss rate.

Comment

The analysis of this test has revealed that if a TG packet is malformed, it is considered as lost. This information has to be taken into account in the different analysis of this Thesis.

7.3.8 Test 8 - Loss of ACKs

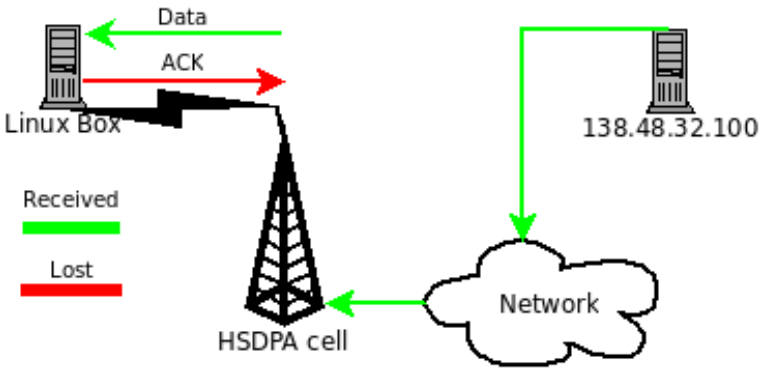


Figure 7.8: Schema of a lost ACK.

The emulation realized on the NT is devoted to the VoIP and the video streaming, which generally work on the UDP protocol. Therefore, the loss of ACK (as detailed on Fig. 7.8) is not important. In the modelling of the testbed, it could take a place in the emulated load of network; but it was not the main objective. Indeed, the 3G technology aiming to the multimedia in the cellular networks, this testbed identifies the trends for the VoIP and the video streaming in a context of different cells, users (static or mobile).

Nevertheless, if the goals change in the future, this restriction, where the ACKs on the uplink are never disturbed, could not be left.

The test has been realized in a similar way to the test (see Section 7.3.7). The use of TG is restricted to WIND which is the only provider that allows using the TCP protocol with TG (see Section 7.4 for the explanation). The TG configuration file for the test is included in the script E.4.1 where the average speed to reach is 384 kb/s. This limit is chosen to avoid to reach the limit of the bandwidth during the traffic generation. Indeed, these ACKs appear in the NT essentially in the load of network. Therefore, it is more relevant to test the loss following a traffic where the objective is not to reach the limit of the network.

Results:

The approach used is to check the number of ACKs in the trace of the source and compare it to the number in the trace of the destination. For each test the result is positive; no ACK has been lost. This involves that the assumption done on the NT could be accepted. However, we have to keep in mind that each test was run only once.

7.4 Issue: the firewall problem

The Vodafone firewall problem was one of the main difficulties we met. The problem has been underlined by the CINI laboratory [47]. If the name of the issue is focused on a firewall, this could also be a Network Address Translation (NAT). We should name it: the middlebox problem. Nevertheless, we keep "Firewall problem" to point the same issue that the CINI laboratory. This problem, once understood, appeared in nearly each experiment; therefore, it is relevant to detail it. In a first time, the problem in general will be explained and the solution used will be presented. Afterwards the solution for TG will be detailed because the generator has been used in subsequent experiments, and also because a modification of source code has been necessary.

7.4.1 Position of the problem

Before the beginning of the explanation, it is important to note that this problem concerns only the Vodafone provider; during our experiments, WIND has never shown any similar behaviour.

Vodafone Firewall, seems to be a good protection for the Vodafone network. Indeed, this structure allows any software to generate an outgoing traffic. In that case, the firewall does not interfere in the connection.

But, inversely (i.e. ingoing traffic), it dos not happen on the same way. It is impossible for an agent from the external world to reach one user in the Vodafone network; the doors are closed and the problem appears. This is exactly that kind of scenario that we need, because the NT is focused on the downlink traffic.

As a first approach, this system could be considered as too closed and prevent all programs to work. Developers know the problem and elaborate techniques to avoid it. For example, all features of Skype can be used on the Vodafone connection. This is possible because such software uses both UDP and TCP traffic and implements some techniques for a firewall traversal [34].

For the tools used in these experiments, it is unfortunately not the same; the programs use only one protocol.

7.4.2 How can we bypass the problem?

In the first time, we can consider scenarios which work and do not work. Following the schema (Fig. 7.9), scenario 1, describing an access from an external user does not work. Scenario 2 shows a Vodafone user who generates traffic to a second user belonging to the external network.

Both scenarios are useful. The first one indicates that there is a system which stops the traffic for one direction (the firewall).

The second one shows, by its completion, that it is possible for an internal user, to create a route to an external user. Therefore, on this system, it manages to define a way from one IP to an other IP following a source port and a destination port.

Once the route is created, the user can generate a traffic and stop it when s/he wants. When the traffic generation stops, the firewall cannot be sure that it is the end of the traffic; it has to wait for a while in order to be certain that the traffic has stopped.

The solution appears at this moment: while the firewall is waiting to close the route, we will generate our traffic in the other sense for the same IPs, same ports but inverting destination and source (this is scenario 3 on Fig. 7.9).

7.4.3 Details of the solution for TG

As we saw before (see Section 7.2.1), TG allows in its configuration file to specify the destination port but not the source port. After reading the source code, it appears that the source port is

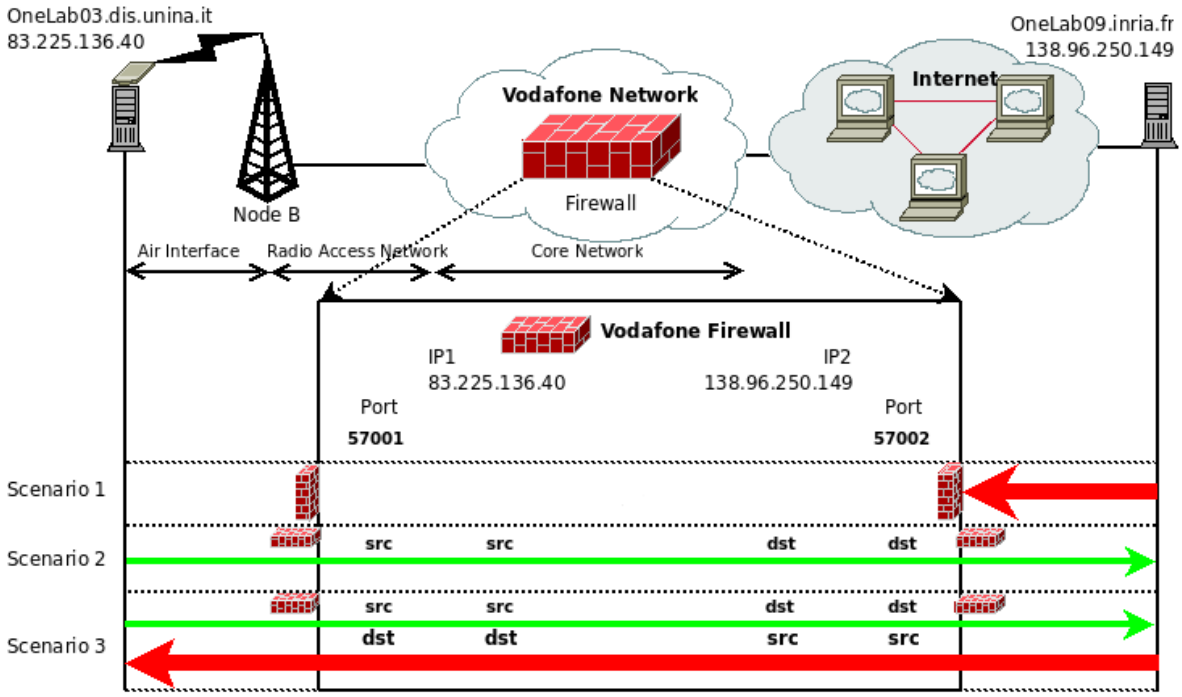


Figure 7.9: Schema of the Vodafone firewall problem.

chosen dynamically. The first modification is then: force the source port of TG (see Appendix C.1.1)[2]. The solution which generates traffic in the other direction as fast as possible creates another problem: if we want to use the same port for two connections on the same machine, and we generate the second connection (client-server) too fast, the operating system forbids it.

Why? Because on a UNIX system, a connection creates a socket which passes through three states: ESTABLISHED, TIMEWAIT, CLOSED. Therefore, as long as the socket is not closed, it has the grip on the port. However, when the traffic is down, the socket stays more or less 60 sec in the TIMEWAIT state. Here comes the difficulty: the firewall closes the route after approximately ten seconds.

The solution is the second modification of the source code: force the new socket to re-use the address (the port) already bound (see appendix C.1.2).

In this configuration, it is now possible to realize scenario 3 of Fig. 7.9. Even if it could be interesting to see how much time the firewall waits before closing, the implementation of the scenario does it as fast as possible (so the answer is unknown). Nevertheless, it is relevant to note that in these conditions, it only takes few seconds to establish the second connection. This time-out can be illustrated with a traffic graph from `wireshark` (Fig. 7.10).

As previously described, the first part of the flow (in green) opens the route on the firewall and then the connection, downlink traffic (in red), is established.

The five seconds of time-out are necessary to close correctly the first connection and also to start the second automatically through `ssh` connections.

[2]The numbers for the ports have been chosen arbitrarily, but without being well-known ports

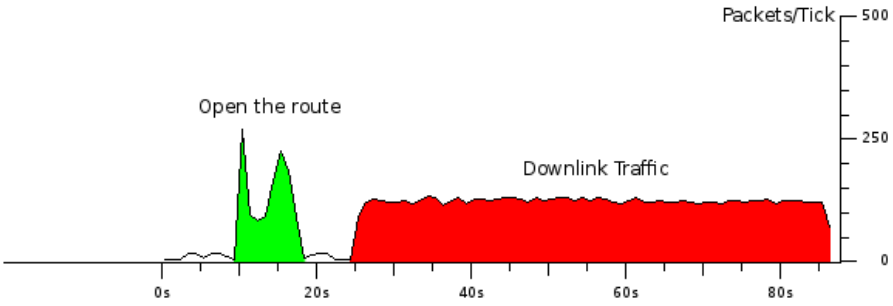


Figure 7.10: openFirewall.

7.4.4 Limits

This scenario solves the problem with the UDP protocol. In the case of TCP the "Downlink traffic" (see Fig. 7.10) never wants to start. Why? It appears that to each end of the "open the route" part, TG loses one packet. This packet loss has as consequence that the second part of the scenario can not start.

It is possible that with another implementation of the scenario, which could try to better manage the ends of connections, both systems could communicate. The time limit after which the firewall closes the route would be important to know.

This was not the objective of this implementation which tries to realize the connection as fast as possible. Furthermore we can keep in mind that WIND has not this problem and therefore it is the reference for the TCP protocol in this environment. In general, multimedia applications (video streaming and VoIP) use UDP which implies that the tests and experiments focus mainly on this protocol. Finally, it is important to note that to deploy this kind of solution, we need an access to the ports of the software; if it is the case for TG, it is not possible for `openRTSP` in case of real traffic (see Section 8.2.2).

7.4.5 Alternative: STUN

As described in Section 7.4.4, the solution on the source and destination ports modification has limits. An alternative that can be taken into account is the Session Traversal Utilities for NAT (STUN) protocol (created in 2003, RFC 3489). STUN protocol allows multimedia applications (data, voice and video) to pass through routers and firewalls, configured on NAT [48]. An example of the use of STUN protocol is shown in Fig 7.11:

Telephone A would like to realize a VoIP session with C (by SIP and RTP). In this configuration, the telephone A has only a private IP address, behind the NAT. It needs to obtain its public address from the NAT. This case involves problems in Session Initiation Protocol (SIP) protocol which needs the public address of the client.

The STUN solution: STUN server B gives to the telephone A its public IP address, and port number that the NAT has allocated for the application's UDP connections to remote hosts. Then the SIP client can integrate the public IP address in its SIP/RTP traffic [48].

It could be interesting to compare this solution with the solution implemented for this Thesis.

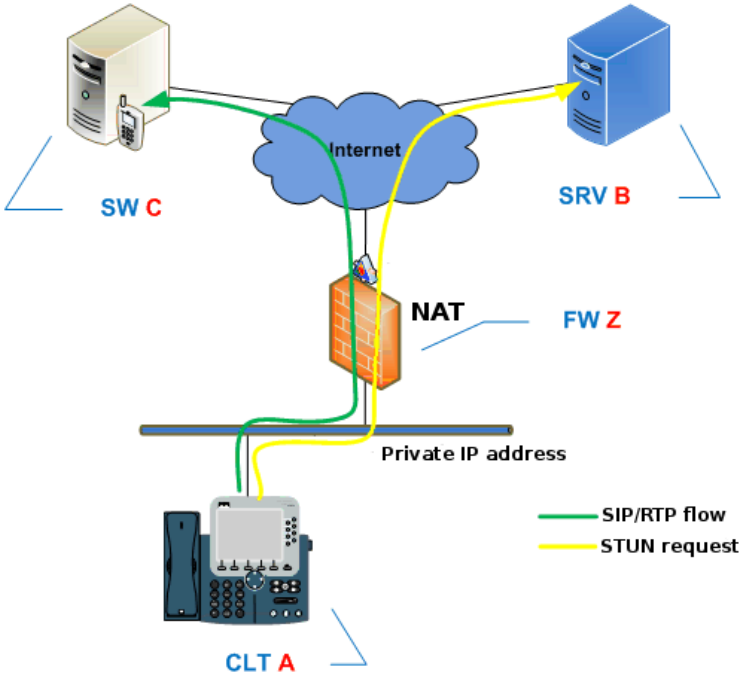


Figure 7.11: STUN protocol example for a SIP/RTP flux [48].

7.5 Results Analysis

7.5.1 Assumptions and consequences

In this chapter different properties have been detailed. Among these characteristics, few of them have not a concrete result. We will discuss them in order to obtain the best description of the environment.

The first elements that we can develop are the traffic classes and the load of network. Both have one common characteristic: there is no technique, no tool in our environment which allows to obtain an information. In that conditions, we can only speculate the values. Concerning the classes, we suppose that it works following the 3GPP specifications adding a QoS sytem. For the load of network, we consider that following the idea that the use of the 3G technology is not very spread yet, we make the assumption that the load of network is low.

Another point that we cannot discover is the RLC mode. The explanation of the test involves the assumption of the easiest solution: the transparent mode.

Afterwards, deduced properties can be revealed. This is the case for the SF and the transport channel. Indeed, both elements are determined by an other characteristic. The result of the first and the seventh tests indicate that HSDPA technology is deployed in the environment.

The consequence of the use of this technology is that both properties are fixed. The SF is equal to 16 and the transport channel uses the HS-DSCH channel (refer to Section 2.7).

Nevertheless all points cannot be revealed with the hardware. For the type of user, the only difference is the type of contract: pre-paid and post-paid.

7.5.2 Physical environment

Following the set of tests, it is interesting to locate as best as possible the physical environment (buildings, cells, distance, terminals). The goal is to assign the providers to the cells available in the surroundings. This approach wants to illustrate that the surroundings must be taken into account, because they have an influence on the results.

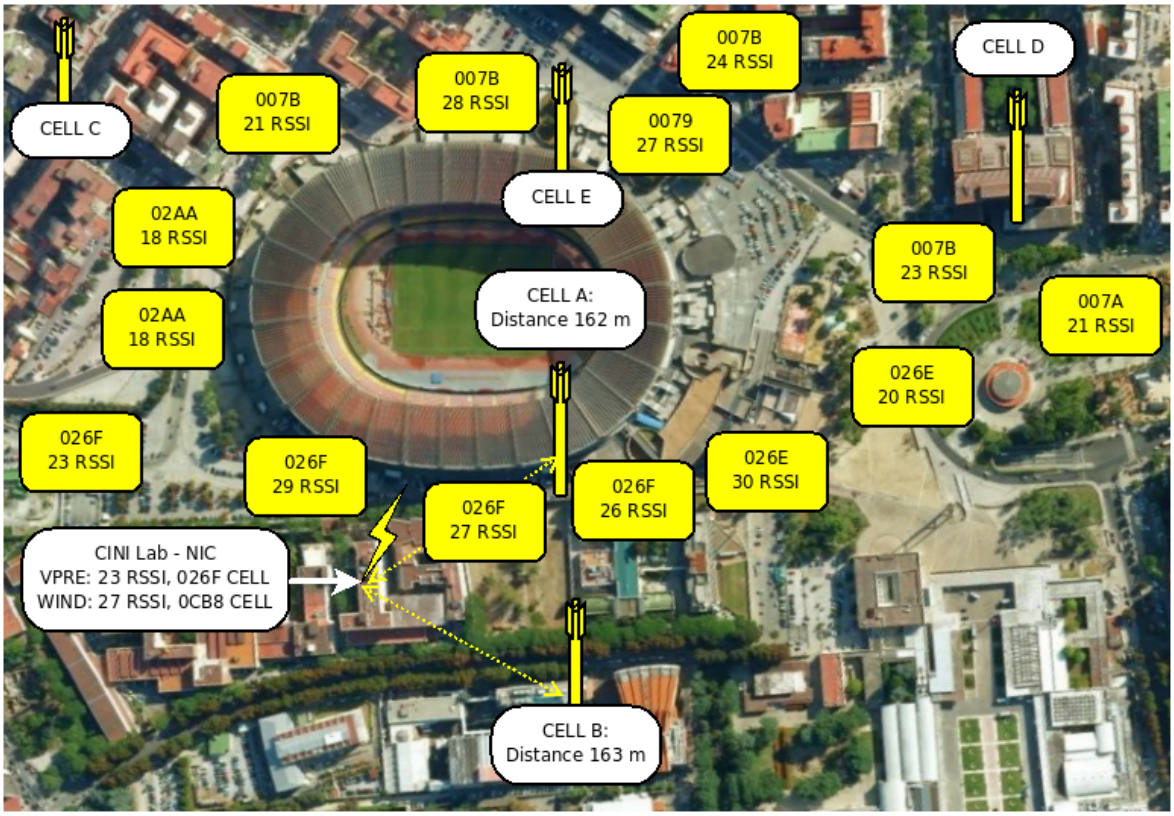


Figure 7.12: Illustration of the mobility experiment.

Fig. 7.12 shows the nearest cells from the laboratory, and also the NIC position. Cells have been pointed by means of `Google Earth`, and followed by a visual confirmation.

Other information provides by a mobility experiment can be accepted. This experiment consists of moving NIC around the physical environment. Data of this experiment have been introduced in Fig. 7.12. The cell ID has been checked, like the signal strength, and is shown on Fig. 7.12.

It could be interesting in this environment to identify which cell matches with Vodafone or WIND. The first element which can be revealed by the experiment is the distance: 162 m from NIC to cell A, 163 m from NIC to cell B (Distances obtains by means of `Google Earth`). We can also point the signal strength: an average of RSSI equal to 23 for Vodafone and 27 for WIND.

In this situation we dispose of two providers and two different cells (A and B). Nevertheless the information collected cannot help to identify the pairs: cell and provider.

In order to determine what cell we exactly use (following the provider), we try to identity it on a theoretical way. In this approach we use the Okumura-Hata model [49, 50]. The idea of the model is that the received power decreases as the distance between the MT and its NodeB increases.

This relation is shown in [1, p.62]. The model identifies the signal strength following different variables: the power of the cell and its height, the height of the terminal and the distance from the cell.

Here is the Okumura-Hata equation:

$$L_U = 69.55 + 26.16 \log f - 13.82 \log h_B - C_H + [44.9 - 6.55 \log h_B] \log d$$

Where $C_H = 3.2(\log(11.75 * h_M))^2 - 4.97$

L_U : Path loss in Urban Areas. Unit: decibel (dB): **has to be computed**.

h_B : Height of base station Antenna. Unit: meter (m): cell A: **25**; cell B: **45** (estimated).

h_M : Height of mobile station Antenna. Unit: meter (m): **12** (estimated).

f : Frequency of Transmission. Unit: megahertz(MHz): reality: 2.1 Ghz but the model limit is **1.92 Ghz**.

C_H : Antenna height correction factor.

d : Distance between the base and mobile stations. Unit: kilometer (km): cell A: **0,162**; cell B: **0.163**.

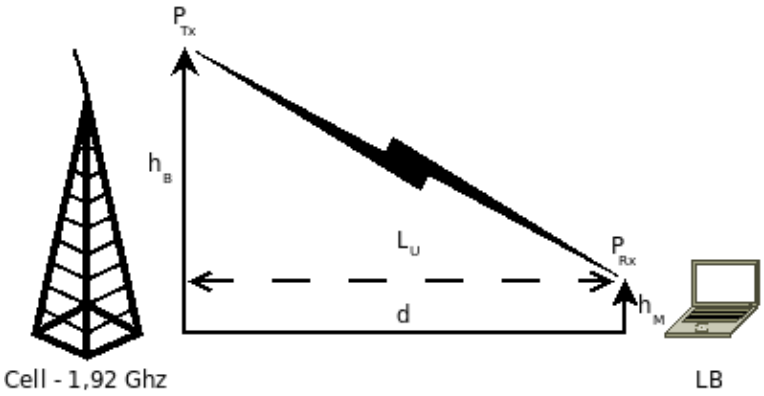


Figure 7.13: Okumura-Hata model elements.

The computed path loss for cell A is 98.05 dB and for B is 95.94 dB. Next it is necessary to link the path loss with the Receiving Power rate (P_{Rx}), as illustrated in Fig. 7.13.

Therefore, to obtain P_{Rx}: $P_{Tx} - L_U = P_{Rx}$, where P_{Tx} means Transmissing Power rate.

Following [3], the maximal value for P_{Tx} is 43 dBm, and if the cell uses a shared transport channel (as supposed in Section 7.5.1), we can consider that 80% of the power can be allocated. Let consider that the connection uses High-Speed Downlink-Shared Channel (HS-DSCH), P_{Tx} is equal to 42 dBm.

In these conditions we obtain a P_{Rx} equals to -56.05 dBm for cell A and -53.04 dBm for cell B. To compare these values with the RSSI that we can measure, we can use Table B.2.2 given in appendix provided by Option manufacturer.

Results:

Cell A: RSSI of 28
Cell B: RSSI of 30

If we consider LB is in better conditions than PRN (see Fig. 6.2), cell B should be associated to the highest RSSI which is WIND. Therefore cell A should be assigned to Vodafone. We can also note a difference between our signal strength results and this model, this can be explained by the fact that our tests have been realized indoor.

Nevertheless, we assume that this approach contains few estimations; it must be considered only as an assumption on the physical environment.

7.5.3 PRN and LB through three providers

In a first time, we compare PRN and LB. In most of case, LB is better than PRN. Is there really a difference or is there a parameter which involves all results? We take in consideration the second one. Indeed, as shown in Fig. 6.2, it appears that PRN is in worst propagation conditions than LB. PRN is in a small room, on the ground, next to the wall; LB is in the middle of a bigger room. This is the only pertinent reason which justify the difference between PRN and LB.

For the providers, it is the same: WIND seems better and VPRE and VPOST similar; does it indicate that WIND has a better technology than Vodafone? Of course not. As the signal strength shows it to us, WIND seems to have a better situation. In this case, it is normal that it has the best results.

These are the only point which differs; otherwise is it similar? The behaviour might change in another environment and create differences between Vodafone and WIND; or between post-paid and pre-paid.

7.5.4 Real environment and NT

As we saw in the different tests, the comparison between the real environment and NT cannot be realised. The environment uses the HSDPA technology which is not suitable with UMTS and NT. In addition such variables are fixed. This is the case of the controlled users: we can manage only one user in the experiments.

Chapter 8

Experiments

This chapter describes the two experiments that were performed in December 2008 at the CINI laboratory in Napoli. The goal was to reproduce the experiments that were performed earlier on the NT by Hugues Van Peteghem. The experiments have been designed as similar as possible to those on the NT. However, because of some constraints due to the real environment, some changes have been applied to the experimental scenarios. They will be described through this chapter.

8.1 Experiments overview

Fig. 8.1 presents an overview of the environment and configuration of the two experiments, video streaming and VoIP, that are described and detailed hereunder.

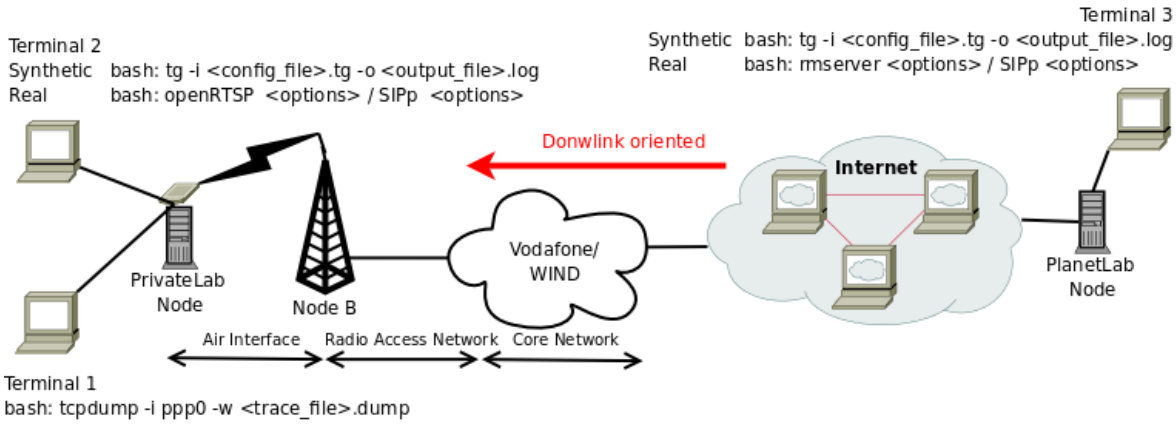


Figure 8.1: Experiments configuration overview.

During the experiments, `tcpdump` was running both on the client side and on the server side. All terminals outputs were redirected into a file to log them.

For both video streaming and VoIP, the experiments are performed using real traffic (with `openRTSP` and `SIPp`) and synthetic traffic (with `TG`).

As for tests described in Chapter 7 and previously explained in Section 7.3, the experiments were run only once by means of the main script (see Appendix E.1 for the script).

8.2 Video Streaming

This section describes the configuration and the tools used to perform the video streaming experiments.

Following 3GPP specifications, the Motion Picture Experts Group Layer-4 (MPEG-4) standard was chosen as video encoding for the IP-based streaming services of UMTS to mobile terminals. It has been chosen as a standard since it can support very low bit rates down to 5 kbps and low frame rates of 15 frames/s [1].

8.2.1 Synthetic Traffic

As the objective of the following section is to detail the synthetic experiment, a generic, three-level traffic model (see Fig. 8.2) has been chosen in the PhD Thesis to characterise the traffic [1]. Here is the meaning of the levels:

Session level : Lasts as long as the application is running. Its statistics are mainly influenced by user behaviour.

Connection level: Describes the connection behaviour of a single session.

Packet Level: Represents the packet inter-arrival and size distribution for each state of the Connection Level.

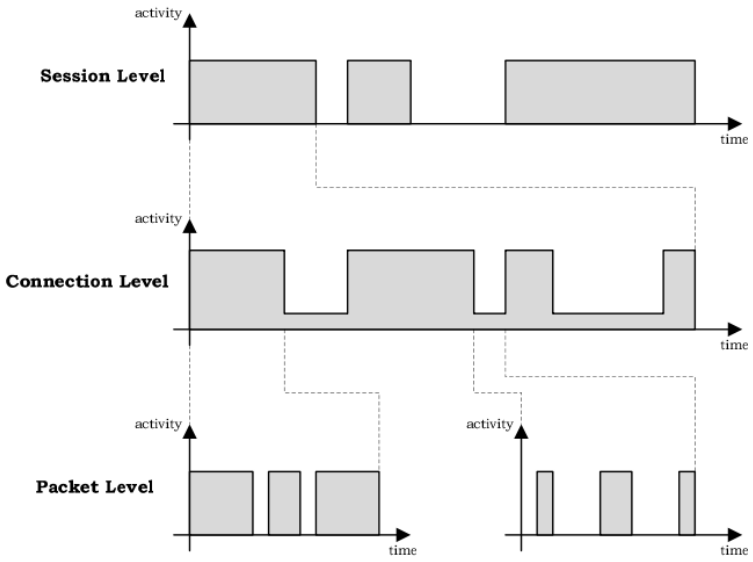


Figure 8.2: Three-level traffic model [1].

According to the multi-level model, a session is equal to a connection since the session lasts during the while streaming connection. The session length is determined by an exponential distribution around 120 seconds.

Video content compression in MPEG-4 standard is done via a Group of Pictures (GoP); either by intra-frame compression removing redundancies within a simple frame of video, or by inter-frame compression removing redundancies across a GoP over time. MPEG-4 uses both intra- and inter- frame compression, and its frames are designated as I-, P- or B- frames.

Considering a constant bit rate video streaming at 25 frames/s, the size of the frames of each video session is described in [51] by a gamma distribution with different parameters. This depends on whether the user is streaming a high or low quality video [1, p.47].

In order to implement this experiment, TG is used. TG configuration takes into account the gamma distribution to determine the packet length. Indeed, the model must assign mean and standard deviation of the size for each frame category. These values have been computed in the PhD Thesis, in order to illustrate a low or medium quality video.

The generator of TG input files for the video streaming emulation is based on the source code used for the PhD Thesis (see Appendix D.2). This generator is managed by the synthetic video script (`videoStreamTG.sh`, in Appendix E.5) realized for LB/PRN environment. `videoStreamTG.sh` is called with the gamma distribution parameters in the main script (see Appendix E.1).

Considering that the length of the session is 120 seconds, it has to be noted that both scenarios (low and medium quality) are realized one by one without respecting any model on the sessions inter-arrivals.

As described in Section 8.1, the synthetic video script is based on the scenario illustrated in Fig. 8.1. This schema identifies the hardware and commands necessary to perform the experiment.

8.2.2 Real Traffic

Server

As described in the PhD Thesis [1], `Helix Server` [52], a multi-format, cross platform streaming server, was used as the video streaming server. The evaluation version for Linux RHEL 4.0 was used and the license was renewed every month.

The streaming server was installed on a PlanetLab Node located in France (`onelab09.inria.fr`). The reason of this choice is the distant location. It should give a clearer impact on the latency than a local location. Also, during the `Helix Server` installation, the PlanetLab Node in Napoli gave us some troubles that remained unexplained despite (limited) investigations.

`Helix Server` was started in the background as root with the following command (from the main `Helix Server` installation directory):

```
./Bin/rmserver rmserver.cfg &
```

Client

As client, `openRTSP` [53] was used. It is a command-line program that can be used to open, stream, receive, and record media streams that are specified by a RTSP URL (i.e., an URL that begins with *rtsp://*). This tool was selected for its useful command-line interface, since no graphical interface is running on PlanetLab Nodes. Moreover, another advantage of this tool is its ability to receive media streams over TCP or over UDP and record them. Also, `openRTSP` provides some QoS statistics.

Videos files

In [1], Hugues Van Peteghem used RealVideo format for the video streaming experiments. However, for our experiments the MPEG-4 format was preferred to RealVideo format for several reasons.

The first motivation is the difficulty to find a command-line multimedia player able to download and record a RealVideo stream. To run the official `RealPlayer`, a graphical interface is needed, but as already mentioned such an interface is missing on PlanetLab. Moreover, at least at the time of the tests design, the Linux version of this player (version 11) could not record the video stream. Only the Windows version was able to record it in a proprietary format of RealNetworks : Internet Video Recording (.ivr) format, hardly convertible into another format. There was no direct converter, at least at that time.

Another reason is the planned post-processing for analysis purposes. To stream a RealVideo file, the proprietary transport protocol Real Data Transport (RDT) is used. Because of this protocol, `Wireshark` [42] can not compute the stream statistics (like jitter and packet loss). It was more complicated to obtain these statistics, given the tight schedule.

The RTP protocol is used to stream MPEG-4 videos. This protocol is more widely used than RDT, more tools can handle with it. Furthermore, the MPEG-4 format is the standard in UMTS for streaming videos.

Table 8.1 gives the properties of the two chosen videos for the video streaming experiments.

Table 8.1: Videos chosen for the video streaming experiments.

	Properties				
Videos	Duration	Frame Rate	Height	Width	Time stamp Frequency
`videoTest-1.mp4`	59 sec	25 fps	240 px	320 px	90,000 Hz
`videoTest-2.mp4`	7 sec	30 fps	240 px	320 px	5,544 Hz

The first video has been chosen for its standard/recommended Sampling Frequency or Time stamp Frequency (i.e. 90,000 Hz for MPEG-4) following the RFC 3016 [54] and the RFC 3640 [55]. This choice is explained in Chapter 9. The second video is a sample video included in the `Helix Server` package. These videos were placed in the sub-directory `Content` of the `Helix Server` directory.

The videos can be found at this link : `http://www.umtsstreamingexperience.be/validation/`.

Tests

Videos were both downloaded once in TCP and once in UDP. However, during the experiments using Vodafone connection (VPOST and VPRE), due to the firewall problem (see Section 7.4), the real video streaming traffic using UDP transport protocol could not pass through the firewall. Contrary to the synthetic traffic (with `TG`) or the real VoIP traffic (with `SIPp`), the trick for the firewall traversal could not be applied since the destination port for the download traffic could not be chosen.

The command used to get a video from the server (and record it) is the following :

`openRTSP -f` *`Framerate`* `-w` *`Width`* `-h` *`Height`* `-4 -n -Q -t -v`
`rtsp://`*`ServerAddress`*`:`*`port`*`/`*`VideoFile`*`.mp4 >` *`received_VideoFile`*`.mp4 2>` *`log.txt`*

where
`-f` is the video frame rate
`-w` is the video image width
`-h` is the video image height
`-4` to output a '.mp4'-format file (to 'stdout')
`-n` to be notified when RTP data packets start arriving
`-Q` to output QoS statistics about the data stream
`-v` to play only the video stream
`-t` to stream RTP/RTCP data over TCP, rather than (the usual) UDP
`>` to record the standard output (i.e., 'stdout') into a file
`2>` to record the standard error output (i.e., 'stderr') into a file

For example, to get `videoTest-1.mp4` using UDP transport mode :

```
openRTSP -f 25 -w 320 -h 240 -4 -n -Q -v
rtsp://onelab09.inria.fr:8554/videoTest-1.mp4 > received_UDP_videoTest-1.mp4
2> log_UDP_videoTest-1.txt
```

Only the video stream was downloaded and recorded (`-v` option) because only video was interesting for our analyses. Also, when downloading both video and audio, the created file of the received video has not always the same size. The cause is presumably due to the `openRTSP` writing process while handling some audio codec which are not well supported.

Recording the received video stream is needed to compare the original video and the received one. The goal is to evaluate the QoE using Peak Signal-to-Noise Ratio (PSNR) metric. The results are given in Chapter 9.

8.3 VoIP

This section describes the configuration and the tools used to perform the VoIP experiments.

To encode voice traffic, the 3GPP has decided to adopt the AMR vocoder, which is a patented audio data compression scheme optimized for speech coding, with a rate of 12.2 kbps [56]. When the user is speaking, the codec produces 32 Byte packets every 20 ms [1].

8.3.1 Synthetic Traffic

This kind of traffic is considered as bidirectional, there is a need for synchronisation. In a real conversation, both participants are speaking throughout the session. Therefore, the emulated downlink and uplink traffics have to be synchronized and mixed [1].

The model, illustrated in Fig. 8.2 (Section 8.2.1), identifies three levels in the traffic: session, connection and packet level. In this type of application, a session starts as long as the user makes a phone call. The NT can emulate several independent session arrivals; nevertheless this experiment aims to identify the characteristics of one session.

The session duration for the VoIP on NT is exponentially distributed with a mean of 120 seconds [57].

Once the session starts, the connection model has to take into account the silence periods in the human voice activity. As the voice activity represents only 32% of the overall time, NT follows the two-state Markovian chain model [56].

When a user is speaking, the codec produces 32-Byte packets every 20 ms. The NT uses TG to emulate this traffic. As described before, this experiment does not show several sessions with a varying inter-arrival time. It realizes two scenarios one by one.

Both scenarios are instances of the two-state Markovian chain model. They vary on the inter-arrivals of packets and on the length of the session. The first one lasts 1'04" and the second 1'26".

The scenarios have been selected in an arbitrary way in the set of scenarios which were provided by the NT. The only condition was they had to last at least one minute.

The configuration of TG for both scenarios can be found in Appendix D.3. This experiment is managed by a script, it is detailed in Appendix E.6; this script is called in the main script (see Appendix E.1).

Finally we have to consider that this experiment happens in a similar way than the generic schema in Fig. 8.1.

8.3.2 Real Traffic

This real traffic VoIP experiment is quite different from the experiment performed on the NT.

On the Namur Testbed, the real traffic VoIP tests were run with two Grandstream Networks BudgeTone-100 Series IP phones [58]. "They used the G.720 vocoder with silent suppression, which is close to the AMR vocoder since it also produces 32 - Byte packets every 20ms." [1]

In our case, because of the environment constraints, the experiments should be designed differently. First, because of the unavailability of these IP phones, a software that can reproduce exactly or as close as possible the same vocoder had to be found. Also, no sound card was available on the machines used for the experiments. It was impossible to use a microphone.

For the VoIP real traffic experiments, `SIPp` [59] was used. It is a free Open Source test tool and traffic generator for the SIP protocol. It can read custom XML scenario files describing from very simple to complex call flows. `SIPp` can also send media (RTP) traffic through pcap replay. Media can be audio or audio and video.

The pcap file was home brewed. The `Asterisk` server [60] was installed on a PlanetLab Node (`onelab09.inria.fr`). The Interactive Voice Response (IVR) of the `Asterisk` server was called via `Ekiga` [61] while running `Wireshark` to capture the RTP stream (the voice as payload). The RTP stream containing the voice is then extracted into a pcap file. The duration of the voice recording is about 120 seconds. By capturing the voice of the IVR, the following constraint is satisfied : the VoIP downlink stream has to represent a real human intervention in a conversation. In other words, it has to respect the stochastic distribution explained in [1, p. 42].

As neither `Ekiga` nor `Asterisk` support AMR vocoder, the GSM 06.10 codec [62, 63] was chosen since it is the closest available codec to AMR vocoder. Because AMR vocoder is patented, it is not implemented and supported out of the box by Open Source softwares like

`Ekiga` and `Asterisk`.

`SIPp` was used on both sides : client and server. As shown in Fig. 8.3, the scenario was designed to have the capacity to pass through a firewall while using UDP transport protocol. A stream is sent in uplink, then in downlink passing by the same port.

The XML scenario files (see appendix E.9) and the pcap files can be found at this link : `http://www.umtsstreamingexperience.be/validation/`.

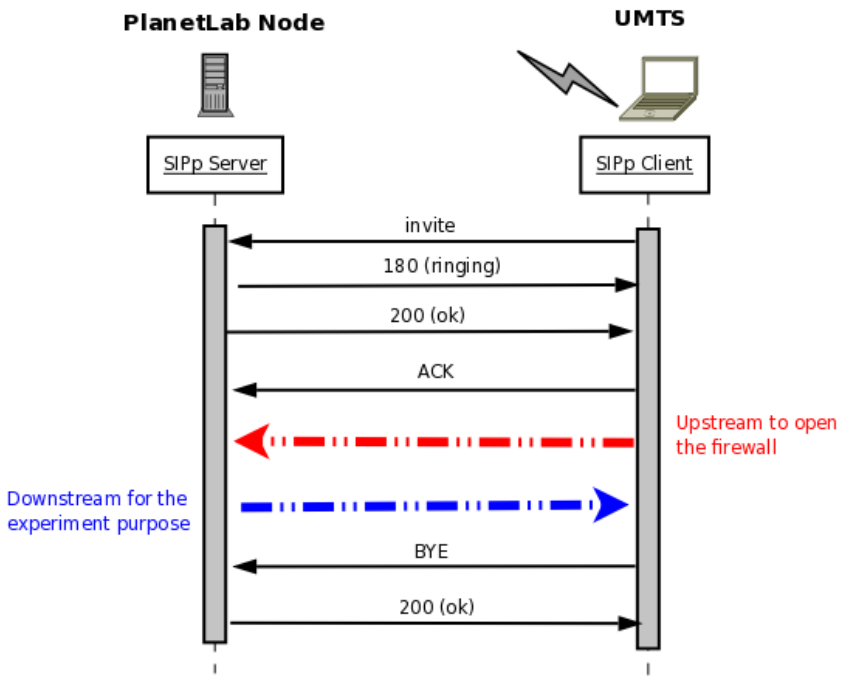


Figure 8.3: SIPp scenario.

To start the `SIPp` server, the following command was used :

`sipp -sf` *`AlternateXMLScenario`* `-i` *`Interface`* `-p` *`Port`* `-mi` *`MediaInterface`* `-mp` *`MediaPort`* `-m` *`Calls`* `-trace_msg -trace_shortmsg -trace_screen -trace_err -trace_stat -trace_counts -trace_rtt -trace_logs`

To start the `SIPp` client, the following command was used :

`sipp` *`ServerAddress:Port`* `-i` *`Interface`* `-p` *`Port`* `-mi` *`MediaInterface`* `-mp` *`MediaPort`* `-sf` *`AlternateXMLScenario`* `-l` *`SimultaneousCalls`* `-r` *`CallRate`* `-m` *`Calls`* `-trace_msg -trace_screen -trace_err -trace_stat -trace_rtt -trace_logs`

where
`-sf` to load an alternate xml scenario file
`-i` to set the local IP address for 'Contact:','Via:', and 'From:' headers
`-p` to set the local port number
`-mi` to set the local media IP address
`-mp` to set the local RTP echo port number

`-m` to stop the test and exit when '*Calls*' calls are processed (in our case, 1)
`-l` to set the maximum number of simultaneous calls (in our case, 1)
`-r` to set the call rate (in calls per seconds) (in our case, 1)
`-trace_msg` to display sent and received SIP messages
`-trace_shortmsg` to display sent and received SIP messages as CSV
`-trace_screen` to dump statistic screens
`-trace_err` to trace all unexpected messages
`-trace_stat` to dump all statistics
`-trace_counts` to dump individual message counts
`-trace_rtt` to allow tracing of all response times
`-trace_logs` to allow tracing of <log> actions

Here is a concrete example of these commands, in the case of an ETH connection (i.e. non-UMTS connection) between a PlanetLab Node and a PrivateLab Node :

On the Server side (PlanetLab Node : `onelab09.inria.fr`) :

```
sipp -sf uas_for_real_voiptest-2minutes.xml -i onelab09.inria.fr -p 9002
-mi onelab09.inria.fr -mp 9902 -m 1 -trace_msg -trace_shortmsg -trace_screen
-trace_err -trace_stat -trace_counts -trace_rtt -trace_logs
```

On the Client side (PrivateLab Node : `onelab03.dis.unina.it`) :

```
sipp onelab09.inria.fr:9002 -i onelab03.dis.unina.it -p 9001 -mi
onelab03.dis.unina.it -mp 9901 -sf uac_for_real_voiptest-2minutes-PR.xml -l 1
-r 1 -m 1 -trace_msg -trace_screen -trace_err -trace_stat -trace_rtt -trace_logs
```

Chapter 9

Analysis of the experiments

In this chapter, we analyse the results of the experiments described in Chapter 8, in terms of classical QoS metrics (jitter and packet loss rate) computed at the application layer. QoE evaluation is also performed for the video streaming by using PSNR metric. These experiments were realized in December 2008 at the CINI laboratory in Napoli.

9.1 Jitter measurement

As the End-to-End delay from the terminal to the RNC value is not an information that is easily to retrieve in a real environment, the E2E delay value, as given in the PhD Thesis [1], is replaced by the jitter metric.

The jitter is a term which is not easy to define because it is used in different ways by different groups of people. Because having many definitions, it can cause confusion.

Following the RFC 3393 [64], "Jitter" commonly has two meanings: "The first meaning is the variation of a signal with respect to some clock signal, where the arrival time of the signal is expected to coincide with the arrival of the clock signal. This meaning is used with reference to synchronous signals and might be used to measure the quality of circuit emulation, for example. There is also a metric called "wander" used in this context.

The second meaning has to do with the variation of a metric (e.g. delay) with respect to some reference metric (e.g. average delay or minimum delay). This meaning is frequently used by computer scientists and frequently (but not always) refers to variation in delay." [64]

The variation can be caused by network congestion, timing drift, or route changes. In multimedia streams, the packet delay variation can be removed by using a play-out buffer at the destination. It is important to know, in this case, the maximum delay variation, which is used to size play-out buffers for such applications. However, for interactive real time applications (i.e. time-sensitive applications) like VoIP, packet delay variation might affect the performance. In audio communications, a high packet delay variation might cause pops and clicks.

In the first definition we will use, jitter refers to the packet delay variation and is calculated as follows (see appendix F.1.2 for our implementation code):

$$Jitter_n = |Delay_n - Delay_{n-1}|$$

where n is the current received packet, and

$$Delay_n = Arrival_n - Departure_n$$

The average jitter is defined as

$$AverageJitter = \frac{\sum_{1}^{n} Jitter}{n}$$

This method is applied to calculate the jitter for all types of traffic (real and synthetic).

The second way of computing the jitter is only used for RTP streams (real video streaming and real VoIP traffic). Actually, it is the interarrival jitter which is different from the jitter previously explained, but it is sometimes also simply called "jitter" in documentation (what causes confusion). It is computed thanks to `Wireshark`. Moreover, for real video streaming, the interarrival jitter is also computed thanks to `openRTSP`. Both `Wireshark` and `openRTSP` calculate the interarrival jitter following the RFC 3550 [65], as explained hereunder.

"The interarrival jitter J is defined to be the mean deviation (smoothed absolute value) of the difference D in packet spacing at the receiver compared to the sender for a pair of packets. As shown in the equation below, this is equivalent to the difference in the "relative transit time" for the two packets; the relative transit time is the difference between a packet's RTP timestamp and the receiver's clock at the time of arrival, measured in the same units.

If S_i is the RTP timestamp from packet i, and R_i is the time of arrival in RTP timestamp units for packet i, then for two packets i and j, D may be expressed as

$$D_{i,j} = (R_j - R_i) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i)$$

The interarrival jitter should be calculated continuously as each data packet i is received from source, using this difference D for that packet and the previous packet $i - 1$ in order of arrival (not necessarily in sequence), according to the formula

$$J_i = J_{i-1} + \frac{|D_{i-1,i}| - J_{i-1}}{16}$$ " [65]

According to the definition above-cited, the computation of the interarrival jitter depends on the RTP timestamp. The latter "reflects the sampling instant of the first octet in the RTP data packet. The sampling instant must be derived from a clock that increments monotonically and linearly in time to allow synchronization and jitter calculations. [..] The clock frequency is dependent on the format of data carried as payload and is specified statically in the profile or payload format specification that defines the format, or may be specified dynamically for payload formats defined through non-RTP means." [65]

In other words, the RTP timestamp which is based on the sampling frequency is used in the interarrival jitter calculation. The sampling frequency depends of the codec. So, the RTP timestamp is based on the sampling frequency of the codec. In the case of MPEG-4 video stream, following RFC 3640 [55], it is recommended that the rate (sampling frequency) be set to 90,000 Hz. This is why the `videoTest-1.mp4` was chosen for the video streaming experiments in addition of the `videoTest-2.mp4` which has a different (non-standard) rate.

If the wrong sampling frequency is used the calculations will give wrong results. This might be the cause of the following problem. In `Wireshark`, the RTP statistics give weird interarrival jitter values for the video streams. For example, it gives a mean jitter of 120 seconds for an experiment which lasts 7 seconds.

Some investigations led us to make a quick little modification in the `Wireshark` (version 1.0.4) source code (see appendix C.3). The modification consists of forcing the clock rate for the statistics computation. From that point, we have two specific versions of `Wireshark`, one with the clock rate set to 90,000 and the other with the clock rate set to 5,544, respectively for `videoTest-1.mp4` and `videoTest-2.mp4`.

Despite this modification, the interarrival jitter values computed by these versions are not exactly the same than those computed by `openRTSP` but they are close.

In the case of `openRTSP`, the source code provides a function to compute the jitter, which is actually the interarrival jitter, and is given in timestamp units. However, the latter is not printed in the QoS statistics (`-Q` option).

Some addition were made in the source code (see appendix C.2) to display the interarrival jitter, and to convert it into time units (in seconds).

Notes

As `Wireshark` (official version) is not giving realistic results for (interarrival) jitter, we only give in the following tables results from the `modified Wireshark`. Also, the number of received packets displayed by `openRTSP` is different (about +/- 10 packets) than the one displayed by `Wireshark`. However, both softwares display 0% of packets lost. Sometimes, `Wireshark` reports packets lost while `openRTSP` reports no packet lost. These observations lead us to keep in mind that tools are not always reliable. From a tool to another, the way of calculating jitter, packet loss, and other statistics might differ. It would explain the difference in the results shown in the following tables.

In addition to `openRTSP` and the `modified Wireshark`, we wrote our own script for computing the jitter (see appendix F.1.1) which is based on the first definition of the jitter previously explained. It computes it from the `tcpdump` captures at both client and server sides. So, we need the capture from both sides to have the departure time and the arrival time of packets.

However, in some cases, the first packets of the stream in the server side capture are missing. The cause seems to be that `tcpdump` does not start quickly enough while the stream has already started. As the RTSP packets which described the payload of the following RTP packets are missing, `Wireshark` can not interpret the RTP payload of the following packets. It identifies them as TCP packets with an unspecified RTSP payload (see Fig. 9.1). This is only in the case of RTP over TCP. For RTP over UDP, the problem does not appear.

Therefore, the RTP sequence number included in the RTP payload is not readable. Since on the client side, we have the full capture, `Wireshark` can interpret RTP payload and give the RTP sequence number and the TCP sequence number. The latter is used to match (pair) packets (those on client side, and those on server side).

Due to this inconvenience, the jitter is calculated only for available packets on both sides. Moreover, the packet loss percentage is not calculated with our own script in case of real traffic analysis. It is only calculated with `openRTSP` and `modified Wireshark`.

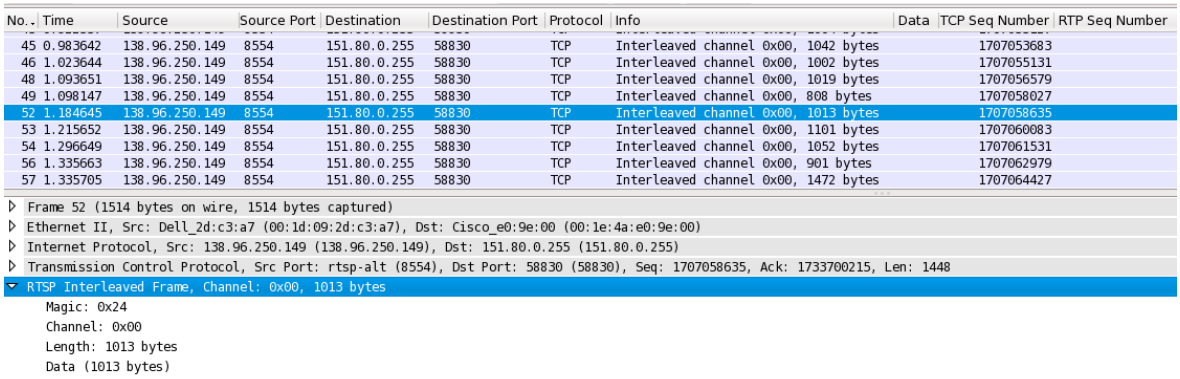


Figure 9.1: Hidden RTP payload in TCP packets.

In the following tables, the "-" symbol means that the test could not be performed due to the firewall issue (see Sections 7.4 and 8.2.2). Graphs in this chapter illustrate the jitter computed by our own script. Concerning the Graphs representing the Cumulative Distribution Function of the jitter, it is always Linux Box which has been selected to illustrate it.

9.2 Video Streaming

9.2.1 Real Traffic

Results

Table 9.1 and Table 9.2 give, respectively, for `videoTest-1.mp4` and `videoTest-2.mp4`, the packet loss results returned by **openRTSP** and *modified Wireshark* (on the client side), following this pattern : **openRTSP** | *modified Wireshark*.

Table 9.1 and especially Table 9.2 illustrate the previously discussed matter of the tools reliability. While `openRTSP` returns 0% of packet lost, `modified Wireshark` returns 99.5% of packet lost for `videoTest-2.mp4`. At the application level, packet loss is 0% since videos were fully and correctly recorded. As shown, the problem appears especially with `videoTest-2.mp4`. The origin of weird results returned by `modified Wireshark` might be the same than the one of wrong jitter calculation, previously discussed. However, the modification of the `Wireshark` source code does not solve the problem for packet loss statistic, contrary to jitter statistic. Obviously, `openRTSP` is here more reliable than `modified Wireshark` and therefore consider a packet loss of 0%.

Table 9.1: `videoTest-1.mp4` - Packet loss.

		Packet Loss [%]			
System	Transport	ETH	VPOST	VPRE	WIND
PrivateLab Node	TCP	**0.0** | *0.0*	**0.0** | *0.4*	**0.0** | *0.3*	**0.0** | *2.3*
	UDP	**0.0** | *0.0*	-	-	**0.0** | *0.0*
Linux Box	TCP	**0.0** | *0.0*	**0.0** | *1.1*	**0.0** | *0.3*	**0.0** | *0.3*
	UDP	**0.0** | *0.0*	-	-	**0.0** | *0.0*

Table 9.2: `videoTest-2.mp4` - Packet loss.

		Packet Loss [%]			
System	Transport	ETH	VPOST	VPRE	WIND
PrivateLab Node	TCP	**0.0** | *0.0*	**0.0** | *99.5*	**0.0** | *99.5*	**0.0** | *99.5*
	UDP	**0.0** | *0.0*	-	-	**0.0** | *0.0*
Linux Box	TCP	**0.0** | *0.0*	**0.0** | *99.6*	**0.0** | *99.5*	**0.0** | *99.5*
	UDP	**0.0** | *0.0*	-	-	**0.0** | *0.0*

For average jitter results shown in Table 9.3 and Table 9.4, the difference between the Ethernet connection (ETH) and UMTS operators (VPOST, VPRE and WIND) is clear. Fig. 9.2 presents the average jitter computed by our own script and Fig. 9.3 shows the Cumulative Distribution Function of the jitter for `videoTest-1.mp4` over TCP on the three providers. The VPOST connection has a worse jitter than VPRE and WIND but the difference is not so big, all the more so the experiments were run only once.

The average interarrival jitter is significantly better over UDP than over TCP. It fits with the transport mode mechanism. UDP does not care about loss or any other traffic issue. It never waits feedback from the destination and thus can send packets regularly. The delay between sending packets is more regular.

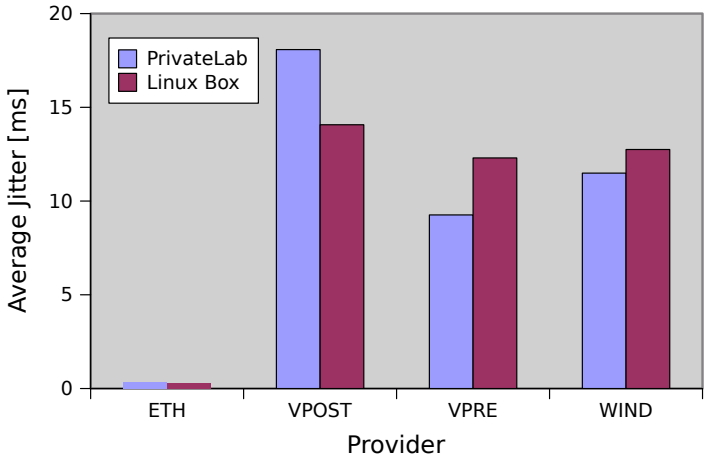


Figure 9.2: Average jitter for `videoTest-1.mp4` over TCP.

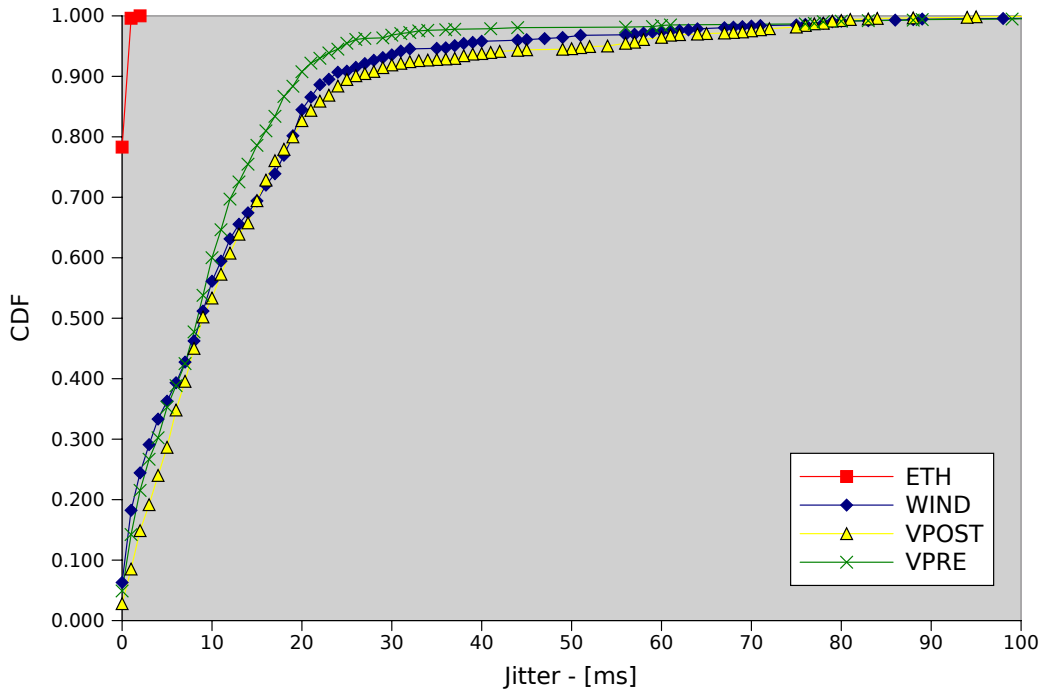


Figure 9.3: Cumulative Distribution Function of the jitter for `videoTest-1.mp4` over TCP.

Table 9.3 and Table 9.4 give, respectively, for `videoTest-1.mp4` and `videoTest-2.mp4`, the average jitter results return by **openRTSP**, *modified Wireshark* (on the client side) and our own script, following this pattern : **openRTSP** | *modified Wireshark* | our own script.

Table 9.3: `videoTest-1.mp4` - Average jitter.

		Average Jitter [ms]			
System	Transport	ETH	VPOST	VPRE	WIND
PrivateLab Node	TCP	**7.74** | *11.68* | 0.32	**52.98** | *31.53* | 18.08	**35.37** | *29.65* | 9.26	**34.04** | *35.91* | 11.49
	UDP	**3.87** | *0.53* | 0.77	-	-	**7.43** | *20.58* | 6.82
Linux Box	TCP	**11.77** | *11.39* | 0.3	**30.92** | *31.12* | 14.07	**32.25** | *32.06* | 12.3	**36.13** | *37.62* | 12.75
	UDP	**1.63** | *0.93* | 0.6	-	-	**6.63** | *14.99* | 6.49

Table 9.4: `videoTest-2.mp4` - Average jitter.

		Average Jitter [ms]			
System	Transport	ETH	VPOST	VPRE	WIND
PrivateLab Node	TCP	**20.38** | *13.56* | 0.43	**41.66** | *18.04* | 15.46	**34.63** | *20.12* | 13.11	**39.14** | *18.18* | 8.5
	UDP	**11.0** | *3.03* | 2.43	-	-	**9.19** | *12.94* | 7.03
Linux Box	TCP	**17.85** | *12.33* | 0.49	**33.91** | *17.24* | 13.9	**35.35** | *18.06* | 7.3	**36.07** | *20.9* | 9.22
	UDP	**3.96** | *1.97* | 0.48	-	-	**6.31** | *8.81* | 7.79

PSNR

The most traditional way to objectively evaluate the quality of a streamed video is the calculation of the Peak Signal-to-Noise Ratio (PSNR) [66] between the original video signal and the received video signal. PSNR is the ratio between the maximum possible power of a signal and the power of degradations that affects the fidelity of its representation. For more details, please refer to [1].

To measure the PSNR, we first have to store the streamed video to compare it with the original one. Then, the two videos have to be synchronized frame-by-frame because the metric compares the videos frame-by-frame. To compute the PSNR, `pnmpsnr` [67] is used. This tool computes the difference between two images. `pnmpsnr` is part of `Netpbm` [68]. `Netpbm` is a toolkit for manipulation of graphic images, including conversion of images between a variety of different formats.

The streamed video was recorded with `openRTSP`. To split up the video into frames, `VirtualDub` [69] was used. This tool is a video capture/processing utility. As it can not handle MPEG-4 video format, videos had to be first encoded into MPEG-1 format. For this processing, `MediaCoder` [70] was used. It is a free universal batch media transcoder.

The frames resulting of the split up processing are encoded into BMP format. As `pnmpsnr` can not handle this format, images had to be converted into ppm format with `bmptoppm` [71].

Then, `pnmpsnr` can be run to analyse videos, frame-by-frame. `pnmpsnr` prints the separate PSNRs of the luminance, and chrominance (Cb and Cr) components of the colors. As in the PhD Thesis, only the PSNR of the luminance is taken.

For the PSNR in lossy image and video compression, typical values are between 30 and 50 dB, whereas a PSNR under 20 dB is considered as unacceptable. Nonetheless, due to the non-linear behaviour of human visual system, PSNR values are not perfectly correlated with a perceived visual quality. [1]

In the following tables, the 'Perfect' column refer to the percentage of perfect frames. These frames are exactly identical in both original and received video and played exactly at the same time. A perfect PSNR is equal to $+\infty$. The 'Accept.' column presents the percentage of acceptable frames in the terms of the PSNR ($\geq$ 20 dB) whereas the 'Unaccept.' column represents the percentage of unacceptable frames ($<$ 20 dB). The 'Mean' column shows the mean PSNR computed on all the frames except the perfect ones.

In some cases, the very last frame of the received video is missing. The precise cause is not really known. Presumably, it seems to be a post-processing problem rather than a real packet loss. One of the possible causes would be a default during the recording (writing file) of the video stream by `openRTSP`. Another cause would be a default during the split up processing by `VirtualDub`. In this last case, repeating the process gives always the same result (i.e. the last frame was missing).

For the PSNR computation, if the last frame of the received video is missing, the last frame of the original video is dropped. In other words, the PSNR computation only takes into account the frames that are available on both videos.

Table 9.5 and Table 9.6 present the obtained results of the PSNR evaluation for, respectively, `videoTest-1.mp4` and `videoTest-2.mp4`.

Table 9.5: PSNR evaluation results - `videoTest-1.mp4`.

			PSNR			
System	Operator	Transport	Perfect [%]	Accept. [%]	Unaccept. [%]	Mean [dB]
PrivateLab Node	VPOST	TCP	100	0	0	$+\infty$
		UDP	-	-	-	-
	VPRE	TCP	100	0	0	$+\infty$
		UDP	-	-	-	-
	WIND	TCP	100	0	0	$+\infty$
		UDP	100	0	0	$+\infty$
	ETH	TCP	100	0	0	$+\infty$
		UDP	100	0	0	$+\infty$
Linux Box	VPOST	TCP	100	0	0	$+\infty$
		UDP	-	-	-	-
	VPRE	TCP	100	0	0	$+\infty$
		UDP	-	-	-	-
	WIND	TCP	100	0	0	$+\infty$
		UDP	100	0	0	$+\infty$
	ETH	TCP	100	0	0	$+\infty$
		UDP	100	0	0	$+\infty$

Table 9.6: PSNR evaluation results - `videoTest-2.mp4`.

			PSNR			
System	Operator	Transport	Perfect [%]	Accept. [%]	Unaccept. [%]	Mean [dB]
PrivateLab Node	VPOST	TCP	90.05	9.95	0	55.19
		UDP	-	-	-	-
	VPRE	TCP	89.69	10.31	0	54.96
		UDP	-	-	-	-
	WIND	TCP	89.69	10.31	0	54.96
		UDP	89.69	10.31	0	54.96
	ETH	TCP	89.69	10.31	0	54.96
		UDP	89.69	10.31	0	54.96
Linux Box	VPOST	TCP	89.64	10.36	0	54.96
		UDP	-	-	-	-
	VPRE	TCP	89.64	10.36	0	54.96
		UDP	-	-	-	-
	WIND	TCP	89.58	10.42	0	54.96
		UDP	89.69	10.31	0	54.96
	ETH	TCP	89.64	10.36	0	54.96
		UDP	89.64	10.36	0	54.96

It is important to keep in mind that the received video is never shown on screen, contrary to the experiments performed on the NT. The stream is immediately recorded by `openRTSP` without display. Therefore, the received video is never distorted by the decoding process before the PSNR evaluation process. However as previously described, the video format is changed (from MPEG-4 to MPEG-1) before the PSNR evaluation process. Watching the received videos for a user subjective evaluation is still possible since the video stream is recorded into a video file.

For `videoTest-1.mp4` (Table 9.5), all frames are perfect. It fits with packet loss results returned by `openRTSP` (i.e. 0%). `videoTest-2.mp4` should have the same results, but it is not the case as shown in Table 9.6. The weird results for `videoTest-2.mp4` are not due to any traffic issue. It seems rather to be a matter of video coding, or post-processing, since the not perfect frames are the same and have the same PSNR values for all tests (even ETH).

9.2.2 Synthetic Traffic

Table 9.7, Table 9.8 and Table 9.9 contain results of the experiment on synthetic video streaming. As described in Section 8.2.1, the experiment groups two scenarios: low and medium quality videos. The difference between both was fixed by the PhD Thesis. In order to identify the difference, the average rate is detailed in Table 9.8.

To understand how the network manages this kind of traffic, the packet loss is shown in Table 9.7. This table indicates that the low quality scenario has no problem while the medium quality has few minor difficulties.

Table 9.7: Synthetic Video Streaming traffic - Packet loss.

		`Packet loss [%]`							
System	Transport	ETH		VPOST		VPRE		WIND	
		LQ	MQ	LQ	MQ	LQ	MQ	LQ	MQ
PrivateLab Node	UDP	0.0	0.0	0.0	0.18	0.0	0.0	0.0	0.12
Linux Box	UDP	0.0	0.0	0.0	0.23	0.0	0.26	0.0	0.0

Table 9.8: Synthetic Video Streaming traffic - Average rate.

		`Average rate [Mbps]`							
System	Transport	ETH		VPOST		VPRE		WIND	
		LQ	MQ	LQ	MQ	LQ	MQ	LQ	MQ
PrivateLab Node	UDP	0.145	0.247	0.150	0.25	0.162	0.363	0.150	0.26
Linux Box	UDP	0.163	0.365	0.150	0.23	0.150	0.26	0.164	0.366

Table 9.9: Synthetic Video Streaming traffic - Average jitter.

		`Average Jitter [ms]`							
System	Transport	ETH		VPOST		VPRE		WIND	
		LQ	MQ	LQ	MQ	LQ	MQ	LQ	MQ
PrivateLab Node	UDP	0.734	0.954	9.907	9.304	9.752	11.841	6.902	5.417
Linux Box	UDP	0.813	0.902	9.856	9.404	9.357	10.152	11.232	7.513

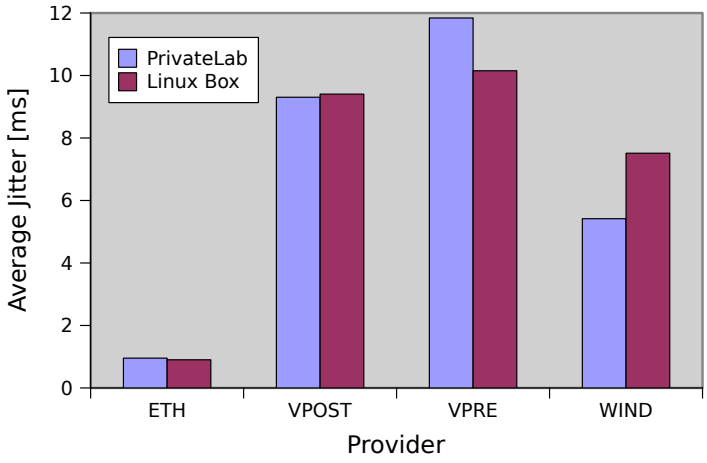


Figure 9.4: Average jitter for the synthetic video streaming experiments (Medium Quality).

Fig. 9.4 illustrates Table 9.9 for the medium quality. We can only note that the different UMTS providers are quite similar. This is also illustrated by Fig 9.5.

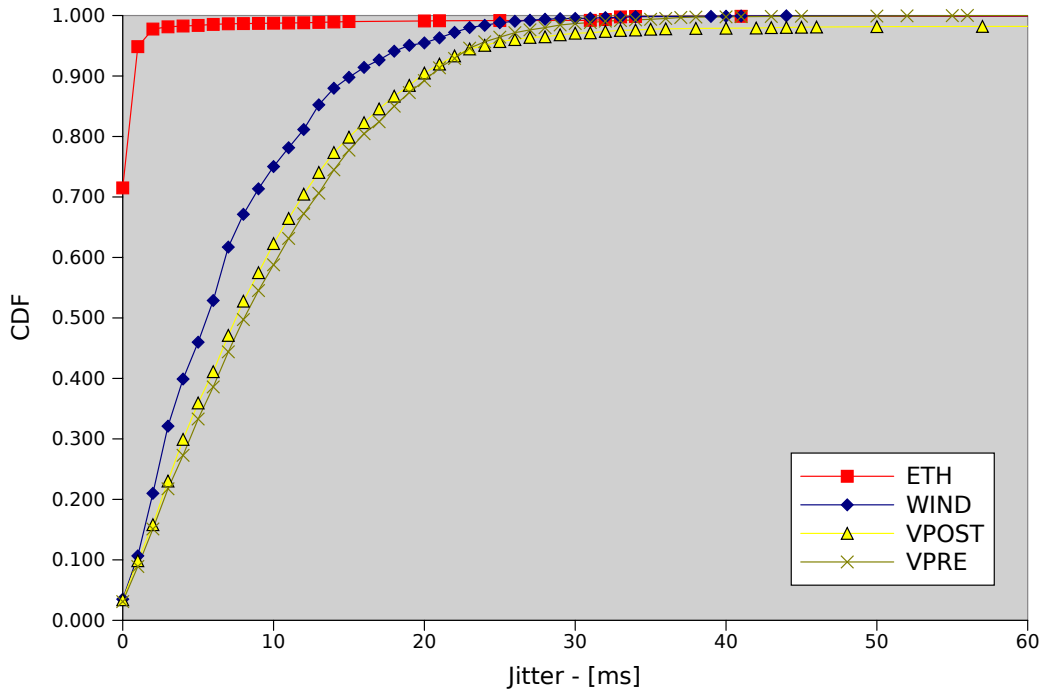


Figure 9.5: Cumulative Distribution Function of the jitter for the synthetic video streaming experiments (Medium Quality).

9.3 VoIP

9.3.1 Real Traffic

Table 9.10 gives the packet loss results returned by *Wireshark* (on the client side) for the UDP stream in downlink, following this pattern : *Wireshark*.

Table 9.10: Real VoIP traffic - Packet loss.

		Packet Loss [%]			
System	Transport	ETH	VPOST	VPRE	WIND
PrivateLab Node	UDP	*0.0*	*0.0*	*0.1*	*0.0*
Linux Box	UDP	*0.0*	*0.0*	*0.0*	*0.0*

Similar to video streaming experiments, packet loss percentage for VoIP experiments is 0% except for one test (0.1%).

Table 9.11 gives the average jitter results returned by *Wireshark* (on the client side) and our own script for the UDP stream in downlink, following this pattern : *Wireshark* | our own script.

Table 9.11: Real VoIP traffic - Average jitter.

		Average Jitter [ms]			
System	Transport	ETH	VPOST	VPRE	WIND
PrivateLab Node	UDP	*4.6* | 0.61	*7.11* | 5.59	*6.49* | 4.82	*8.06* | 6.53
Linux Box	UDP	*4.6* | 0.54	*8.12* | 6.63	*6.23* | 4.61	*5.94* | 4.38

The difference between the Ethernet connection (ETH) and UMTS operators (VPOST, VPRE and WIND) in terms of average jitter is clearer with results returned by our own script than those returned by `Wireshark`. It is important to remind that Wireshark computes the interarrival jitter while our own script computes the jitter (see Section 9.1). Therefore, results show that the packets are regularly sent but it is the delay which is longer in case of UMTS connection.

Fig. 9.6 illustrates the average jitter computed by our own script and Fig. 9.7 shows the Cumulative Distribution Function of the jitter for the real VoIP experiments.

Similar to the PSNR calculation for streamed video, objective quality assessment also exist for audio [1]. Two of them are Perceptual Evaluation of Speech Quality (PESQ) [72] and Perceptual Evaluation of Audio Quality (PEAQ) [73]. The first one is an enhanced perceptual quality measurement for voice quality in telecommunications. The other one is specifically developed to apply to E2E voice quality testing under real network conditions like in VoIP. These objective audio quality assessment are not handled in this dissertation.

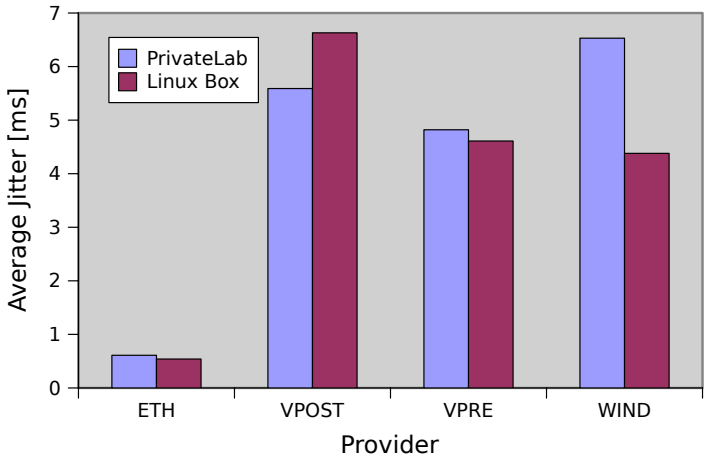


Figure 9.6: Average jitter for the real VoIP experiments.

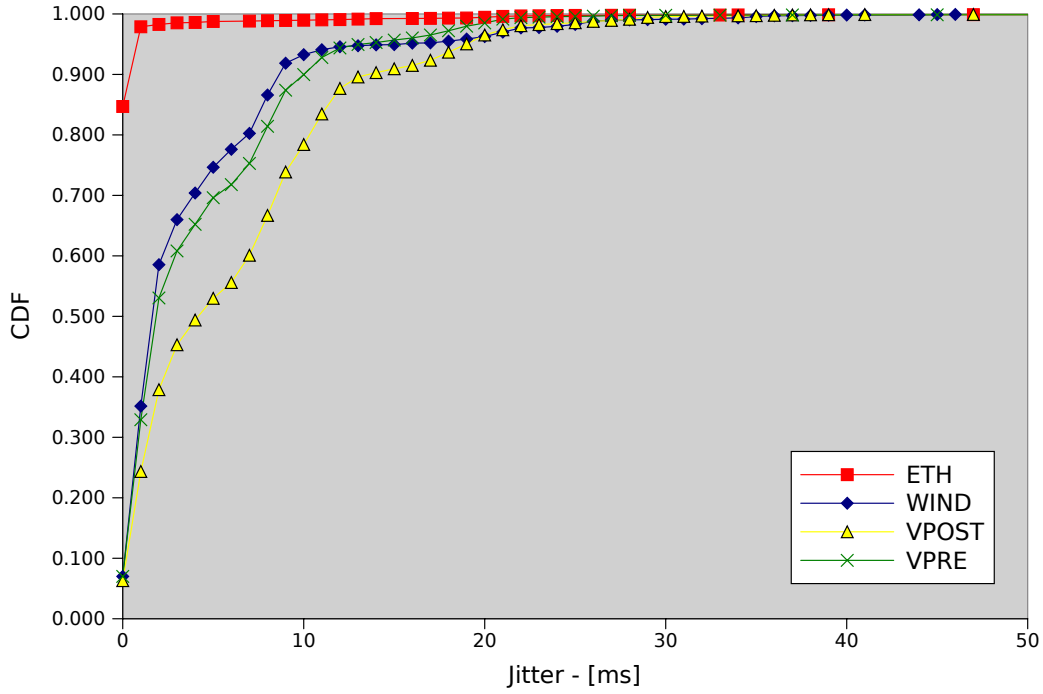


Figure 9.7: Cumulative Distribution Function of the jitter for the real VoIP experiments.

9.3.2 Synthetic Traffic

Table 9.12 and Table 9.13 give the characteristics of this experiment. Following all the results for the packet loss, we can consider that the VPRE and WIND loss appear due to a fortuitous event.

Table 9.12: Synthetic VoIP traffic - Packet loss.

		Packet loss [%]							
System	Transport	ETH		VPOST		VPRE		WIND	
		1	2	1	2	1	2	1	2
PrivateLab Node	UDP	0.0	0.0	0.0	0.0	2.643	0.0	0.0	0.0
Linux Box	UDP	0.0	0.0	0.0	0.0	0.0	0.0	5.163	1.599

Table 9.13: Synthetic VoIP traffic - Average rate.

		Average Rate [%]							
System	Transport	ETH		VPOST		VPRE		WIND	
		1	2	1	2	1	2	1	2
PrivateLab Node	UDP	0.009	0.011	0.009	0.011	0.009	0.011	0.009	0.011
Linux Box	UDP	0.009	0.011	0.009	0.011	0.009	0.011	0.009	0.011

In general no difficulty has been identified as the average rate can confirm it.

Table 9.14: Synthetic VoIP traffic - Average jitter.

		Average Jitter [ms]							
System	Transport	ETH		VPOST		VPRE		WIND	
		1	2	1	2	1	2	1	2
PrivateLab Node	UDP	0.303	0.376	2.257	4.211	7.087	3.016	5.950	4.506
Linux Box	UDP	0.285	0.252	2.087	3.592	9.504	2.888	13.161	6.150

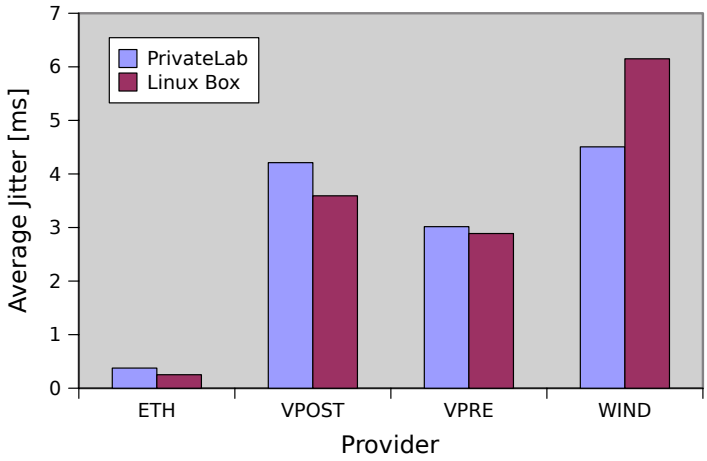


Figure 9.8: Average jitter for the second synthetic VoIP experiments.

One more time, Fig. 9.8 and Fig. 9.9 indicates that providers and systems are similar.

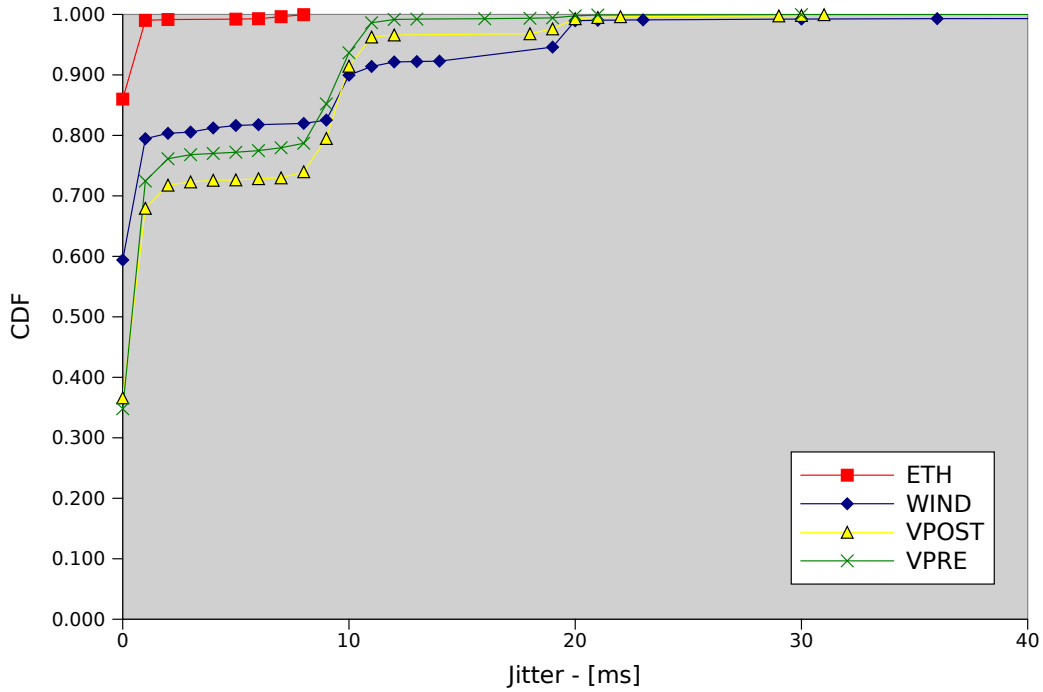


Figure 9.9: Cumulative Distribution Function of the jitter for the second synthetic VoIP experiments.

9.4 Global analysis

When analysing the experimental results, the difference between Linux Box and PrivateLab Node is not as clear as for the environment tests (as explained in Section 7.5.3). Actually, experiments do not push the UMTS connection to the limit. Video streaming and VoIP do not use large bandwidth. Therefore, even if PrivateLab Node is in a worse location than Linux Box, the results are very close.

Moreover, experiments which were first designed for a UMTS Release 99 connection have good results in our case since we use a UMTS Release 5 connection (i.e. HSDPA). These results show that HSDPA gives good results for (light) multimedia applications, at least from a static perspective. The experimental results do not really show a clear difference between operators.

Finally, it is important to remind that all experiments were run only once, as explained in Section 7.3. Therefore, results can not be interpreted as statistically significant and final conclusions can not be drawn. Moreover, as explained in Section 9.1, despite our attempt (i.e. modification of the source code), the results for the average interarrival jitter returned by `openRTSP` and `modified Wireshark` are too scattered. Nothing allows us to choose which one of the two should be used as reference. In order to compare results from synthetic traffic to those from real traffic, we have written our own script for jitter computation. It serves as reference in this dissertation.

Chapter 10

Conclusion

The 3^{rd} Generation of mobile phones is now widespread and the user number is increasingly growing. To test their new algorithms and configurations, operators and infrastructure manufacturers can perform tests in a real environment or perform them in a testbed. The second solution is easier and cheaper than the first one.

Hugues Van Peteghem, as a part of his PhD Thesis at the University of Namur, has designed and implemented an emulating UMTS testbed (the NT). The main objective of this Master Thesis was to validate it by reproducing experiments into a real environment in order to compare results.

As explained, because of discrepancies in the intended real environment, we had to perform tests and experiments in a uncontrolled environment wherein some parameters remained unknown. Moreover, the real environment was a UMTS Release 5 one, whereas the NT complies with UMTS Release 99.

Therefore, we had to perform tests and experiments from a basic end user perspective. After giving the theoretical basis about studied aspects, the experimental methodology in a such environment was described through this dissertation.

The CINI laboratory, where our traineeship and the experiments were carried out, being responsible of the deployment and the integration of the UMTS connection on a PlanetLab Node, tests and experiments were both performed from a PlanetLab environment (using a PlanetLab Node) and from a classical environment (using a common Linux-based laptop), in order to compare results between the two environments. The results show that there is no noteworthy difference between them.

This Master Thesis points out that it is not easy to discover cellular network parameters from a basic end user perspective (i.e. with classical hardware). As explained, TEMS can help in this approach, but were not available at the time of our experiments executions.

Also, experimental results were quite good mainly thanks to the UMTS Release 5 conditions with better performance than the UMTS Release 99. Results show that users have the opportunity to enjoy (light) multimedia applications without being annoyed as long as they stay into a Rel'5 environment.

The presented results in this document are so not directly comparable with those from the Namur Testbed. However, as the NT is bound to evolve to UMTS Release 5, this dissertation can be easily used to reproduce experiments into the future NT. Furthermore, this document is presenting an approach for performing tests and experiments in a real environment from a basic end user perspective and can so be easily used for future work.

Future work

In addition to reproduce experiments into the future NT, it would be interesting to reproduce them in mobile conditions in a real environment. Indeed, this dissertation is only presenting experiments and their results from a static perspective. Perform them in mobile conditions would allow to observe the handover behaviour and compare it with the NT one. Also, tests would be performed at constant as well as varying speeds.

Bibliography

[1] Hugues Van Peteghem. *Building a Testbed Emulating Cellular Networks; Design, Implementation, Cross-Validation and Exploitation of a Real-Time Framework to Evaluate QoS and QoE in the UTRAN.* PhD thesis, Namur University, February 2008. `http://www.info.fundp.ac.be/~hvp/rech/doc/index_en.html`.

[2] Benjamin Evrard and Gille Gomand. Stage Naples 2008 - Rapport d'activités, 2009. FUNDP - Faculté d'Informatique.

[3] Harri Holma and Antti Toskala. *WCDMA for UMTS - HSPA evolution and LTE.* Wiley inter-science, 4^{th} edition, 2007.

[4] UMTS World. `http://www.umtsworld.com`, last visited: August 18, 2009.

[5] Jaana Laiho, Achim Wacker, and Tomas Novosad. *Radio Network Planning and Optimisation for UMTS.* Wiley inter-science, second edition, 2006.

[6] Technical Specification Group Radio Access Network. Multiple Input Multiple Output in UTRA. Technical Report TR 25.876 V7.0.0, 3^{rd} Generation Partnership Project (3GPP), March 2007.

[7] Technical Specification Group Radio Access Network. Feasibility study for Orthogonal Frequency Division Multiplexing (OFDM) for UTRAN enhancement. Technical Report TR 25.892 V6.0.0, 3^{rd} Generation Partnership Project (3GPP), June 2004.

[8] Technical Specification Group Radio Access Network. Delay Budget within the Access Stratum (Release 4). Technical Report TR 25.853 V4.0.0, 3^{rd} Generation Partnership Project (3GPP), 2001.

[9] Harri Holma and Antti Toskala. *WCDMA for UMTS.* Wiley inter-science, 3^{rd} edition, October 2004.

[10] Paul McKenney, Dan Lee, and Barbara Denny. Traffic Generator Software Release Notes. Technical report, SRI International and USC/ISI Postel Center for Experimental Networking, January 2002. `http://www.postel.org/tg/tg.htm`, last visited: August 18, 2009.

[11] Technical Specification Group Radio Access Network. Multiple Input Multiple Output in UTRA. Technical Report TR 25.876 V7.0.0, 3^{rd} Generation Partnership Project (3GPP), March 2007.

[12] Alexandrosz Burulitisz, Sándor Imre, and Sándor Szabó. On the Accuracy of Mobility Modelling in Wireless Networks. In *Proceedings of the 39st International Conference on Communications (ICC)*, Paris (France), June 2004.

[13] Ben Liang and Zygmunt Haas. Predictive Distance-Based Mobility Management for PCS Networks. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom)*, pages 1377–1384, New York (NY - USA), March 1999.

[14] Femtocell. `http://www.thinkfemtocell.com/`, last visited: August 18, 2009.

[15] CellTrack. `http://www.afischer-online.de/sos/celltrack/#newos`, last visited: August 18, 2009.

[16] W. Karner and M. Rupp. Measurement based Analysis and Modelling of UMTS DCH Error Characteristics for Static Scenarios. In *Proc. of 8th International Symposium on DSP and Communication Systems (DSPCS'2005) & 4th Workshop on the Internet, Telecommunications and Signal Processing (WITSP'2005)*, 2005.

[17] W. Karner and O. Nemethova and M. Rupp. A Measurement Based Model for UMTS DL DCH Dynamic Bearer Type Switching. In *International Symposium on Wireless Pervasive Computing 2006*, 2006.

[18] Marilena P. Kallenou Charalampos N. Pitas, Eleni D. Avgeri and Philip Constantinou. Measurements and QoS Analysis of Live-World Mobile Telecommunication Networks. 2008.

[19] Manuel Alvarez-Campana, Enrique Vázquez, Joan Vinyes, and Víctor A. Villagrá. Measuring quality of experience of internet access over hsdpa. 2008.

[20] A. Barbuzzi, F. Ricciato, and G. Boggia. Discovering Parameter Setting in 3G Networks via Active Measurements. *Communications Letters, IEEE*, 2008.

[21] PlanetLab Central. `http://www.planet-lab.org`, last visited: August 18, 2009.

[22] Roberto Canonico. From PlanetLab to PlanetLab Europe. Technical report, University of Napoli Federico II, Laboratorio ITeM-CINI, July 2008.

[23] PlanetLab Europe. `http://www.planet-lab.eu`, last visited: August 18, 2009.

[24] Giovanni Di Stasi Antonio Pescape Giorgio Ventre Alessio Botta, Roberto Canonico. Providing UMTS connectivity to PlanetLab nodes. Technical report, University of Napoli Federico II, Laboratorio ITeM-CINI, December 2008.

[25] Alessio Botta. Planetlab Tutorial. Technical report, University of Napoli Federico II, Laboratorio ITeM-CINI, July 2008.

[26] Orbit. `http://www.orbit-lab.org`, last visited: August 18, 2009.

[27] Emulab. `http://www.emulab.net`, last visited: August 18, 2009.

[28] Vini. `http://www.vini-veritas.net`, last visited: August 18, 2009.

[29] Roberto Canonico. PlanetLab Europe Access Policies. Technical report, University of Napoli Federico II, Laboratorio ITeM-CINI, July 2008.

[30] Serge Fdida. An Open Federated Laboratory to evaluate the possible futures of the Internet. Technical report, Université Pierre et Marie Curie de Paris, Laboratoire LIP6, May 2008.

[31] Susanna Avéssta. OneLab - Future Internet Testbeds. Technical report, FIREworks, December 2008.

[32] OneLab. `http://www.onelab.eu`, last visited: August 18, 2009.

[33] Serge Fdida. Report on Onelab/2 Activities. Technical report, Université Pierre et Marie Curie de Paris, Laboratoire LIP6, April 2008.

[34] Roberto Canonico, Alessio Botta, Giovanni Di Stasi, Alessandro Amirante, Lorenzo Miniero, Salvatore D'Antonio, Simon Pietro Romano, Antonio Cimmino, Bruno Giulio Misculin, and Matteo Dell'Orto. UMTS Gateway. Technical report, Cini Item Laboratory, 2008.

[35] Roberto Canonico, Alessio Botta, Maria Barone, and Salvatore D'Antonio. UMTS Node. Technical report, Cini Item Laboratory, 2007.

[36] Nozomi driver. `http://www.pharscape.org`, last visited: October 27, 2008.

[37] Technical Specification Group Core Network and Terminals. *AT command set for User Equipment (UE) (Release 8)*. 3^{rd} Generation Partnership Project, 2008.

[38] Wvdial Configuration. `http://linux.die.net/man/5/wvdial.conf`, last visited: August 18, 2009.

[39] AT commands of an Option card. `http://support.option.com/support/faq.php?do=article&articleid=67`, last visited: October 27, 2008.

[40] GNU Wget. `http://www.gnu.org/software/wget/`, last visited: August 18, 2009.

[41] Craig Leres Van Jacobson and University of California Berkeley CA Steven McCanne, all of the Lawrence Berkeley National Laboratory. Tcpdump - dump traffic on a network. Technical report, October 2008. `http://www.tcpdump.org`, last visited: August 18, 2009.

[42] Gerald Combs. Wireshark - network protocol analyzer. `http://www.wireshark.org`, last visited: August 18, 2009.

[43] UMTSmon. `http://umtsmon.sourceforge.net`, last visited: August 18, 2009.

[44] KmlCellID.kml. `http://cellid.telin.nl:8080/wasp/jsp/CellStats.jsp`, last visited: August 18, 2009.

[45] Google Earth. `http://earth.google.com/intl/fr/index.html`, last visited: August 18, 2009.

[46] TEst Mobile System. `http://www.ascom.com/en/index/products-solutions/technology-platforms/platform/tems/solutionloader.htm`, last visited: August 18, 2009.

[47] Giovanni Di Stasi. Wireless extensions to the PlanetLab infrastructure. Technical report, University of Napoli Federico II, Laboratorio ITeM-CINI, July 2008.

[48] Entête STUN. `http://www.architoip.com/entete-stun/`, last visited: August 18, 2009.

[49] A. Medeisis and A. Kajackas. On the use of the universal Okumura-Hata propagation predictionmodel in rural areas, 2000.

[50] John S Seybold. *Introduction to RF propagation*. John Wiley and Sons, 2005.

[51] Michael Frey and Son Nguyen-Quang. A Gamma-Based Framework for Modeling Variable-Rate MPEG Video Sources: the GOP GBAR Model. Technical report, IEEE/ACM Transactions on Networking, December 2000.

[52] E. Regan. Helix Server Unlimited, March 2007. `http://www.realnetworks.com/products/media_delivery.html`, last visited: August 18, 2009.

[53] Ross Finlayson. OpenRTSP. `http://www.live555.com/openRTSP/`, last visited: August 18, 2009.

[54] Y. Kikuchi, T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata. RTP Payload Format for MPEG-4 Audio/Visual Streams. RFC 3016 (Proposed Standard), November 2000. `http://www.ietf.org/rfc/rfc3016.txt`, last visited: August 18, 2009.

[55] J. van der Meer, D. Mackie, V. Swaminathan, D. Singer, and P. Gentric. RTP Payload Format for Transport of MPEG-4 Elementary Streams. RFC 3640 (Proposed Standard), November 2003. `http://www.ietf.org/rfc/rfc3640.txt`, last visited: August 18, 2009.

[56] Technical Specification Group Radio Access Network. Feasibility Study for Enhanced Uplink for UTRA FDD (Release 6). Technical Report TR 25.896 V6.0.0, 3^{rd} Generation Partnership Project (3GPP), April 2004.

[57] Technical Specification Group Radio Access Network. Ip transport in utran (release 5). Technical Report TR 25.933 V5.4.0, 3^{rd} Generation Partnership Project (3GPP), January 2004.

[58] John Huang. User Manual, BudgeTone-100 Series IP Phone. Technical Report 1.0.8.32, Grandstream Networks, September 2006. `http://www.grandstream.com/bt101.html`, last visited: October 9, 2008.

[59] Richard Gayraud, Olivier Jacques, and Charles P. Wright. SIPp. `http://sipp.sourceforge.net`, last visited: August 18, 2009.

[60] Mark Spencer. Asterisk. `http://www.asterisk.org`, last visited: August 18, 2009.

[61] Ekiga. `http://ekiga.org`, last visited: August 18, 2009.

[62] Digital cellular telecommunications system (Phase 2+) (GSM); Full rate speech; Transcoding (GSM 06.10 version 5.2.1 Release 1996) . Technical Report ETS 300 961, European Telecommunications Standards Institute (ETSI), January 2001. `http://www.etsi.com`, last visited: August 18, 2009.

[63] H. Schulzrinne and S. Casner. RTP Profile for Audio and Video Conferences with Minimal Control. RFC 3551 (Standard), July 2003. `http://www.ietf.org/rfc/rfc3551.txt`, last visited: August 18, 2009.

[64] C. Demichelis and P. Chimento. IP Packet Delay Variation Metric for IP Performance Metrics (IPPM). RFC 3393 (Proposed Standard), November 2002. `http://www.ietf.org/rfc/rfc3393.txt`, last visited: August 18, 2009.

[65] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003. Updated by RFC 5506. `http://www.ietf.org/rfc/rfc3550.txt`, last visited: August 18, 2009.

[66] Wang, Zhou and Sheikh, Hamid R. and Bovik, Alan C. *Handbook of Video Databases: Design and Applications*, chapter 41 - Objective Video Quality Assessment, pages 1041–1078. CRC Press, September 2003.

[67] Bryan Henderson. pnmpsnr. `http://netpbm.sourceforge.net/doc/pnmpsnr.html`, last visited: August 18, 2009.

[68] Bryan Henderson. Netpbm. `http://netpbm.sourceforge.net`, last visited: August 18, 2009.

[69] VirtualDub. `http://www.virtualdub.org`, last visited: August 18, 2009.

[70] MediaCoder. `http://mediacoder.sourceforge.net`, last visited: August 18, 2009.

[71] Bryan Henderson. bmptoppm. `http://netpbm.sourceforge.net/doc/bmptoppm.html`, last visited: August 18, 2009.

[72] ITU-T Recommendation. Perceptual Evaluation of Speech Quality (PESQ): An Objective Method for End-to-End Speech Quality Assessment of Narrow-band Telephone Networks and Speech Codecs. Technical Report P.862, International Telecommunication Union, 2001.

[73] Opticom. PEAQ — Perceptual Evaluation of Audio Quality. Technical report, Opticom, 1999.

[74] UMTS streaming experience - validation. `http://www.umtsstreamingexperience.be/validation/`, last visited: August 18, 2009.

Appendix A

UMTS connection

A.1 Script to initiate the connection on a Linux box

This script can be found at [74].

```
#!/bin/bash
#umtsInit.sh
echo "##### umtsInit #####"
echo "loading module"
insmod /lib/modules/2.6.18-1.2798.fc6/kernel/drivers/pci/hotplug/nozomi.ko

echo "initialising umts card - 3G mode"
gcom -d /dev/noz0 3G
wait

echo "registration..."
gcom -d /dev/noz0
wait
```

A.2 Script to realize the connection UMTS

This script can be found at [74].

```
#!/bin/bash
#umtsStart.sh
# input 1 = provider
# input 2 = directory

echo "##### variables configuration #####"
PROVIDER=$1
DIRECTORY=$2

echo "##### umtsStart #####"
if [ "$DIRECTORY" = "root" ];
then
        # we use the Linux Box
        sh /$DIRECTORY/fundp/umtsInit.sh
        echo "Try to connect on UMTS"
        wvdial pcmcia $PROVIDER &
else
        # we use the privateLab node and its own scripts...
```

```
19          su -c "umts start"
20  fi
21
22  echo "Wait few seconds for the connection..."
23  sleep 30
24
25  STR=`ifconfig ppp0`
26  STR2="ppp0"
27  if [ ${STR:0:4} = $STR2 ];
28  then
29          IPUMTS=`ifconfig ppp0 | sed -n 's/.*inet adr:\([^.]*.[^.]*.[^.]*.[^ ]*\).*
                $/\1/p'`
30          echo "##### Connected - $IPUMTS #####"
31  else
32          echo "##### ppp0 doesn't exist #####"
33  fi
34
35  echo "##### add the IP of the PlanetLab node (inria) on ppp0 #####"
36  if [ "$DIRECTORY" = "root" ];
37  then
38          # we use the Linux box
39          su -c "route add 138.96.250.149 ppp0"
40  else
41          # we use the privateLab node and its own scripts...
42          su -c "umts add 138.96.250.149 "
43  fi
```

A.3 Connection established

```
##### umtsStart #####
##### umtsInit #####
loading module
initialising umts card - 3G mode
Set 3G only mode
registration...
SIM ready
Waiting for Registration..(120 sec max)
Registered on Home network: "vodafone IT",2
Signal Quality: 22,99
Try to connect on UMTS
Wait few seconds for the connection...
--> WvDial: Internet dialer version 1.54.0
--> Cannot get information for serial port.
--> Initializing modem.
--> Sending: ATZ
ATZ
OK
--> Sending: ATZ
ATZ
OK
--> Sending: ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
OK
--> Sending: AT+CGDCONT=1,"IP","web.omnitel.it"
AT+CGDCONT=1,"IP","web.omnitel.it"
OK
--> Modem initialized.
--> Sending: ATD*99***1#
```

```
--> Waiting for carrier.
ATD*99***1#
CONNECT 1800000
--> Carrier detected.  Starting PPP immediately.
--> Starting pppd at Thu Dec 11 16:59:48 2008
--> pid of pppd: 2830
--> Using interface ppp0
--> pppd: Modem
--> pppd: Modem
--> pppd: Modem
--> pppd: Modem
--> pppd: Modem
--> pppd: Modem
--> local  IP address 83.225.136.40
--> pppd: Modem
--> remote IP address 10.64.64.64
--> pppd: Modem
--> primary   DNS address 83.224.65.134
--> pppd: Modem
--> secondary DNS address 83.224.66.134
--> pppd: Modem
##### Connected -  #####
##### add the IP of the PlanetLab node (inria) on ppp0 #####
```

Appendix B

AT-commands scripts

The following scripts are called in a main script (see E.2) and can be found at [74].

B.1 HSDPA

This command can be used to see whether a HSDPA call is in progress. Status will be set to 1 only when a HSDPA transaction is in progress (i.e., when a HS-DSCH transport channel is active), not when merely the cell supports HSDPA.

```
opengt
 set com 115200n81
 set senddelay 0.05
 waitquiet 2 0.5
 let c=0
:hsdpa
 waitquiet 1 0.1
 send "AT_OHCIP?^m"
 get 2 "^m" $s
 get 2 "^m" $s
 let a=len($s)
 let a=a-8
 let $s=$right($s,a)
 if $s <> "0,0" goto hsdpacont
 if c > 3 goto hsdpaexit
 let c=c+1
 pause 1
 goto hsdpa
:hsdpaexit
 print "Error for hsdpa!"
 exit 1
:hsdpacont
 print "HSDPA call in progress?:",$s,"\n"
 waitquiet 1 0.1
 exit 0
```

B.2 Signal quality

B.2.1 +CSQ

This command returns the Received Signal Strength Indication (RSSI):

```
:signal\ n \
waitquiet 1 0.1\ n \
send \ "AT+CSQ\^m\ " \ n \
```

B.2.2 RSSI - dBm

The following table has been provided by Option manufacturer [39] for the "GlobeTrotter HSDPA '7.2 Ready' E" Option card.

RSSI	dBm	RSSI	dBm	RSSI	dBm
0	<-113	11	-91	22	-69
1	-111	12	-89	23	-67
2	-109	13	-87	24	-65
3	-107	14	-85	25	-63
4	-105	15	-83	26	-61
5	-103	16	-81	27	-59
6	-101	17	-79	28	-57
7	-99	18	-77	29	-55
8	-97	19	-75	30	-53
9	-95	20	-73	31	>-51
10	-93	21	-71	99	unknown

B.3 Cell ID

B.3.1 +CREG

This command returns the status of result code presentation. It shows whether the network has currently indicated the registration of the MT and gives two location information elements: local area code and cell id<lac>.

```
opengt
 set com 115200n81
 set senddelay 0.05
 waitquiet 2 0.5
 let c=0
:cellidG
 waitquiet 1 0.1
 send "AT+CREG=2^m"
 waitquiet 1 0.1
 send "AT+CREG?^m"
 get 2 "^m" $s
 get 2 "^m" $s
 let a=len($s)
 let a=a-6
 let $s=$right($s,a)
```

```
16  if $s <> "0,0" goto cellidGcont
17  if c > 3 goto cellidGexit
18  let c=c+1
19  pause 1
20  goto cellidG
21 :cellidGexit
22  print "Error for the cellidG!"
23  exit 1
24 :cellidGcont
25  print "CellId and the area:",$s,"\n"
26  waitquiet 1 0.1
27  exit 0
```

B.3.2 +CGREG

This command is similar to the first one but for a GPRS context.

```
1  opengt
2   set com 115200n81
3   set senddelay 0.05
4   waitquiet 2 0.5
5   let c=0
6  :cellidU
7   waitquiet 1 0.1
8   send "AT+CGREG=2^m"
9   waitquiet 1 0.1
10  send "AT+CGREG?^m"
11  get 2 "^m" $s
12  get 2 "^m" $s
13  let a=len($s)
14  let a=a-8
15  let $s=$right($s,a)
16  if $s <> "0,0" goto cellidUcont
17  if c > 3 goto cellidUexit
18  let c=c+1
19  pause 1
20  goto cellidU
21 :cellidUexit
22  print "Error for the cellidU!"
23  exit 1
24 :cellidUcont
25  print "CellId UMTS and the area:",$s,"\n"
26  waitquiet 1 0.1
27  exit 0
```

B.4 Type of the user

B.4.1 +CAEMLPP

The execute command is used to change the default priority level of the user in the network. If the user does not have subscription for the requested priority level an ERROR result code is returned.

```
1  opengt
2   set com 115200n81
3   set senddelay 0.05
4   waitquiet 2 0.5
```

```
 let c=0
:caemlpp
 waitquiet 1 0.1
 send "AT+CAEMLPP?^m"
 get 2 "^m" $s
 get 2 "^m" $s
 let a=len($s)
 #let a=a-6
 let $s=$right($s,a)
 if $s <> "0,0" goto caemlppcont
 if c > 3 goto caemlppexit
 let c=c+1
 pause 1
 goto caemlpp
:caemlppexit
 print "Error for caemlpp!"
 exit 1
:caemlppcont
 print "CAEMLPP?:",$s,"\n"
 waitquiet 1 0.1
 exit 0
```

B.4.2 +CPPS

This command returns a priority subscription of the user stored on the SIM card. If no explicit priority level subscription is stored on the SIM card, the result code OK is returned.

```
opengt
 set com 115200n81
 set senddelay 0.05
 waitquiet 2 0.5
 let c=0
:cpps
 waitquiet 1 0.1
 send "AT+CPPS?^m"
 get 2 "^m" $s
 get 2 "^m" $s
 let a=len($s)
 #let a=a-6
 let $s=$right($s,a)
 if $s <> "0,0" goto cppscont
 if c > 3 goto cppsexit
 let c=c+1
 pause 1
 goto cpps
:cppsexit
 print "Error for cpps!"
 exit 1
:cppscont
 print "CPPS?:",$s,"\n"
 waitquiet 1 0.1
 exit 0
```

B.4.3 +CFCS

The set command is used to edit the status of the priority level for fast call set-up stored on the SIM card. If the user has no subscription to the priority level status he wants to edit, an

ERROR result code is returned.

```
opengt
 set com 115200n81
 set senddelay 0.05
 waitquiet 2 0.5
 let c=0
:cfcs
 waitquiet 1 0.1
 send "AT+CFCS?^m"
 get 2 "^m" $s
 get 2 "^m" $s
 let a=len($s)
 #let a=a-6
 let $s=$right($s,a)
 if $s <> "0,0" goto cfcscont
 if c > 3 goto cfcsexit
 let c=c+1
 pause 1
 goto cfcs
:cfcsexit
 print "Error for cfcs!"
 exit 1
:cfcscont
 print "CFCS?:",$s,"\n"
 waitquiet 1 0.1
 exit 0
```

Appendix C

Programs modifications

The programs modifications can be found at [74].

C.1 TG

C.1.1 force the source port in prot_udp.c

```
156 /* Allow to reuse the port before the RTT delay */
157 optval = 1;
158 if (setsockopt(sfd,SOL_SOCKET,SO_REUSEADDR,&optval,sizeof(optval))== -1){
159         perror("ReUse_the_socket");
160     exit(errno);
161 }
```

C.1.2 re-use the socket in prot_udp.c

```
162 /* Try to force the source port */
163 sin.sin_family = AF_INET;
164 sin.sin_port = htons(57001);
165 sin.sin_addr.s_addr = INADDR_ANY;
166 if (bind(sfd,(struct sockaddr *)&sin,sizeof(sin))== -1){
167         perror("bind()");
168         exit(errno);
169 }
```

C.2 openRTSP

C.2.1 modifications in RTPSource.cpp

```
24 //addition for Jitter calculation
25 #include <iostream>
26 //end of addition
```

```
212 //addition for Jitter calculation
213   packet_num = 0;
214   last_d = 0;
215 //end of addition
```

```
314 //addition for Jitter calculation
315   int jit = d - last_d;
316   if (jit < 0) jit = -jit;
317   last_d = d;
318   std::cerr << "Packet Number : " << packet_num << "\n";
319   std::cerr << "jit (in timestamp units) : " << jit << "\n";
320   std::cerr << "jit (in time units) : " << jit/(double)timestampFrequency << "\n";
321   std::cerr << "timestampFrequency : " << timestampFrequency << "\n";
322   std::cerr << "fJitter float : " << fJitter << "\n";
323   std::cerr << "fJitter float/timestampFrequency : " << fJitter/timestampFrequency
        << "\n";
324   std::cerr << "fJitter integer : " << (unsigned)fJitter << "\n";
325   std::cerr << "fJitter integer/timestampFrequency : " << (unsigned)fJitter/(
        double)timestampFrequency << "\n\n";
326   packet_num += 1;
327 //end of addition
```

```
405 //addition for Jitter calculation
406 double RTPReceptionStats::jitter_float() const {
407   return fJitter;
408 }
409 //end of addition
```

C.2.2 modifications in RTPSource.hh

```
191 //addition for Jitter calculation
192   double jitter_float() const;
193 //end of addition
```

```
242 //addition for Jitter calculation
243   int packet_num;
244   int last_d;
245 //end of additon
```

C.2.3 modifications in playCommon.cpp

```
1157 //addition for Jitter calculation
1158   *env << "jitter (in seconds) for RDT stream\t" << rdt->jitter()/rdt->
         timestampFrequency << "\n";
1159 //end of addition
```

```
1175 //addition for Jitter calculation
1176   double freq = (double)(src->timestampFrequency());
1177   *env << "Jitter float (in timestamp units) :\t" << stats->jitter_float() << "\n"
         ;
1178   *env << "Jitter integer (in timestamp units) :\t" << stats->jitter() << "\n";
1179   *env << "Timestamp Frequency :\t" << src->timestampFrequency() << "\n";
1180   *env << "Jitter based on integer (in seconds)\t" << stats->jitter()/freq << "\n"
         ;
1181   *env << "Jitter based on float (in seconds)\t" << stats->jitter_float()/freq <<
         "\n";
1182 //end of addition
```

C.3 Wireshark

C.3.1 modifications in tap-rtp-common.c for `videoTest-1.mp4`

```
448 //modification for Jitter calculation
449                 //clock_rate = get_clock_rate(statinfo->pt);
450                 clock_rate = 90000;
451         }else{ /* dynamic PT */
452                 if ( rtpinfo->info_payload_type_str != NULL )
453                         //clock_rate = get_dyn_pt_clock_rate(rtpinfo->
                                info_payload_type_str);
454                         clock_rate = 90000;
455                 else
456                         //clock_rate = 1;
457                         clock_rate = 90000;
458 //end of modification
```

C.3.2 modifications in tap-rtp-common.c for `videoTest-2.mp4`

```
448 //modification for Jitter calculation
449                 //clock_rate = get_clock_rate(statinfo->pt);
450                 clock_rate = 5544;
451         }else{ /* dynamic PT */
452                 if ( rtpinfo->info_payload_type_str != NULL )
453                         //clock_rate = get_dyn_pt_clock_rate(rtpinfo->
                                info_payload_type_str);
454                         clock_rate = 5544;
455                 else
456                         //clock_rate = 1;
457                         clock_rate = 5544;
458 //end of modification
```

Appendix D

TG configuration files

D.1 Simple example

```
#UDP CLIENT                                          #UDP SERVER
on 0:20 udp 192.168.0.239.4322                       on 0:15 udp 192.168.1.64.4322 server
at 5 setup                                           at 1.1 wait 30
at 6 arrival constant 0.01 length constant 450
time 15
```

In this case, the server starts, listens and finishes after 30 seconds. The client starts after 20 seconds and sends packet of 450 bytes every ms; it stops after 15 seconds.

D.2 Synthetic video streaming

This script can be found at [74].

```
1  ////////////////////////////////////////////////////////////////////////
2  //                                                                    //
3  // streaming.c                                                        //
4  //                                                                    //
5  // Implementation of a generator of tg input file for the video       //
6  // streaming simulation.                                              //
7  //                                                                    //
8  //
           //
9  //                                                                    //
10 ////////////////////////////////////////////////////////////////////////
11
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <math.h>
15 #include <time.h>
16 #include <limits.h>
17 #include "mcsim/sim/random.h"
18
19 /* streaming                                                          */
20 /* function which defined packets configuration for TG                */
21 int streaming(int timeTraffic,double ui,double si,double up, double sp, double ub,
       double sb){
22 double scale;
23 double shape;
24
25 double randNumber;
```

```
int i;
int nbpackets;
for(i = 0;i<timeTraffic;i++){
        // Parameters to obtain the frame size of B-frame
        shape = (ub * ub)/(sb*sb);
        printf("shape : %d\n",(int)shape);
        scale = (sb*sb)/ub;
        randNumber=GGammaRandom(shape,scale);
        printf("B%d-frame size:%d\n",i,(int)randNumber);
        // Parameters to obtain the frame size of P-frame
        shape = ((up-ub)*(up-ub))/((sp*sp)-(sb*sb));
        printf("shape : %d\n",(int)shape);
        scale = ((sp*sp)-(sb*sb))/(up-ub);
        randNumber=randNumber+GGammaRandom(shape,scale);
        printf("P%d-frame size:%d\n",i,(int)randNumber);
        // Parameters to obtain the frame size of I-frame
        shape = ((ui-up)*(ui-up))/ ((si*si)-(sp*sp));
        printf("shape : %d\n",(int)shape);
        scale = (si*si)-(sp*sp)/(ui-up);
        randNumber=randNumber+GGammaRandom(shape,scale);
        printf("P%d-frame size:%d\n",i,(int)randNumber);
        nbpackets=floor(randNumber);
        printf("arrival 0.040 length %d\n",nbpackets);
}
return(0);
}

/* Main                                                                    */
/* creation of the tg input file.                          */
/* input 1 : IP                                                            */
/* input 2 : port                                                          */
/* input 3 : TimeTraffic                                       */
/* input 4 : ui                                                            */
/* input 5 : si                                                            */
/* input 6 : up                                                            */
/* input 7 : sp                                                            */
/* input 8 : ub                                                            */
/* input 9 : sb                                                            */

int main(int argc, char *argv[]){
/* Variables configuration                                                 */
char * ip=argv[1];
char * port=argv[2];
int timeTraffic= atoi(argv[3]);

/* Creation of basic lines                                                 */
printf("# Configuration scripts for a VideoStreaming session\n");
printf("on 0:03 udp %s.%s tos 39 \n",ip,port);
printf("setup\n");

/* Creation of streaming lines                                             */
streaming(timeTraffic,strtod(argv[4],NULL),strtod(argv[5],NULL),strtod(argv[6],
    NULL),strtod(argv[7],NULL),strtod(argv[8],NULL),strtod(argv[9],NULL));
return(0);
}
```

D.3 Synthetic VoIP

This script can be found at [74].

```
#!/bin/bash
# createFilesVOIP.sh
# input 1 = directory
# input 2 = IPUMTS

DIRECTORY=$1
IPUMTS=$2

echo "
# Configuration scripts for a VoIP session
#########################################
on 0:01 udp $IPUMTS.57001 tos 39
setup
arrival 0.020000 length 32
packet 42
at 2.440000 arrival 0.020000 length 32
packet 8
at 3.280000 arrival 0.020000 length 32
packet 55
at 4.900000 arrival 0.020000 length 32
packet 15
at 5.740000 arrival 0.020000 length 32
packet 4
at 6.180000 arrival 0.020000 length 32
packet 36
at 7.680000 arrival 0.020000 length 32
packet 73
at 10.540000 arrival 0.020000 length 32
packet 41
at 13.040000 arrival 0.020000 length 32
packet 33
at 16.240000 arrival 0.020000 length 32
packet 84
at 19.980000 arrival 0.020000 length 32
packet 75
at 21.600000 arrival 0.020000 length 32
packet 18
at 25.280000 arrival 0.020000 length 32
packet 32
at 31.240000 arrival 0.020000 length 32
packet 45
at 39.080000 arrival 0.020000 length 32
packet 65
at 42.700000 arrival 0.020000 length 32
packet 70
at 48.060000 arrival 0.020000 length 32
packet 32
at 50.120000 arrival 0.020000 length 32
packet 114
at 57.960000 arrival 0.020000 length 32
packet 5
at 59.700000 arrival 0.020000 length 32
packet 16
at 62.340000 arrival 0.020000 length 32
packet 8
at 64.440000 arrival 0.020000 length 32
```

```
packet 78" > /$DIRECTORY/fundp/voip/synthetic/voipClient1.tg

echo "
# Configuration scripts for a VoIP session
#########################################
on 0:01 udp $IPUMTS.57001 tos 39
setup
at 0.920000 arrival 0.020000 length 32
packet 249
at 8.380000 arrival 0.020000 length 32
packet 25
at 12.900000 arrival 0.020000 length 32
packet 28
at 13.519999 arrival 0.020000 length 32
packet 35
at 15.019999 arrival 0.020000 length 32
packet 5
at 16.339999 arrival 0.020000 length 32
packet 155
at 24.300002 arrival 0.020000 length 32
packet 25
at 25.580000 arrival 0.020000 length 32
packet 21
at 26.999998 arrival 0.020000 length 32
packet 42
at 33.400000 arrival 0.020000 length 32
packet 30
at 37.120001 arrival 0.020000 length 32
packet 2
at 39.139998 arrival 0.020000 length 32
packet 1
at 40.200003 arrival 0.020000 length 32
packet 32
at 41.419997 arrival 0.020000 length 32
packet 158
at 47.539999 arrival 0.020000 length 32
packet 28
at 48.179999 arrival 0.020000 length 32
packet 33
at 51.400000 arrival 0.020000 length 32
packet 26
at 53.419997 arrival 0.020000 length 32
packet 88
at 56.380003 arrival 0.020000 length 32
packet 22
at 57.020003 arrival 0.020000 length 32
packet 37
at 58.700003 arrival 0.020000 length 32
packet 127
at 64.160002 arrival 0.020000 length 32
packet 24
at 64.679999 arrival 0.020000 length 32
packet 6
at 65.599997 arrival 0.020000 length 32
packet 15
at 67.080000 arrival 0.020000 length 32
packet 42
at 69.020003 arrival 0.020000 length 32
packet 58
at 71.800002 arrival 0.020000 length 32
```

```
117 packet␣2
118 at␣73.440001␣arrival␣0.020000␣length␣32
119 packet␣26
120 at␣74.099997␣arrival␣0.020000␣length␣32
121 packet␣16
122 at␣75.400000␣arrival␣0.020000␣length␣32
123 packet␣7
124 at␣77.220000␣arrival␣0.020000␣length␣32
125 packet␣59
126 at␣80.239996␣arrival␣0.020000␣length␣32
127 packet␣40
128 at␣83.000006␣arrival␣0.020000␣length␣32
129 packet␣2
130 at␣83.799994␣arrival␣0.020000␣length␣32
131 packet␣17
132 at␣84.539999␣arrival␣0.020000␣length␣32
133 packet␣13
134 at␣86.799994␣arrival␣0.020000␣length␣32
135 packet␣5" > /$DIRECTORY/fundp/voip/synthetic/voipClient2.tg
```

Appendix E

Scripts for the tests and experiments

E.1 Main script

For each test, this script is modified, the parameters are adapted to the provider, the system, etc.

```
1  #!/bin/bash
2  # test.sh
3
4  echo "##### Variables configuration #####"
5  DIRECTORY="home/unina_umts"
6  INTERFACE="eth0"
7  PROVIDER="ETH"
8  HOUR="18"
9  DATE="23-01"
10 NODEPL="onelab09.inria.fr"
11 IPDISPLAY=127.0.0.1
12
13 echo "##### tests.sh $HOUR - $DATE - $PROVIDER - $INTERFACE #####"
14
15 if [ "$INTERFACE" = "ppp0" ]; then
16      sh /$DIRECTORY/fundp/umtsStart.sh $PROVIDER $DIRECTORY
17      wait
18 else
19      echo "eth0 is used..."
20 fi
21
22 echo "##### affiche heure #####"
23 date
24
25 echo "##### Start atCommands #####"
26 sh /$DIRECTORY/fundp/environnement/atCommands/atCommands.sh $HOUR $DATE $DIRECTORY
       $PROVIDER $INTERFACE
27
28 echo "##### start signal #####"
29 xterm -display $IPDISPLAY:0 -e ssh -t -i ~/.ssh/root_ssh_key.rsa root@onelab03.dis
       .unina.it "sh /root/fundp/signal.sh" &
30 PIDSIG=$!
31 sleep 4
32
33 echo "##### affiche heure #####"
34 date
35
```

```
36 echo "##### Start Wget #####"
37 sh /$DIRECTORY/fundp/environnement/maxDownlink/wget/maxDownWget.sh $HOUR $DATE
       $INTERFACE $DIRECTORY $PROVIDER $NODEPL
38
39 echo "##### affiche heure #####"
40 date
41
42 #echo "##### Start TG 384 - 1450 #####"
43 #sh /$DIRECTORY/fundp/environnement/maxDownlink/tg/maxDownTG.sh $HOUR $DATE udp
       $INTERFACE $DIRECTORY $NODEPL $PROVIDER 1450 0.030 384 $IPDISPLAY
44
45 echo "##### affiche heure #####"
46 date
47
48 #echo "##### Start TG 384 - 48 #####"
49 #sh /$DIRECTORY/fundp/environnement/maxDownlink/tg/maxDownTG.sh $HOUR $DATE udp
       $INTERFACE $DIRECTORY $NODEPL $PROVIDER 48 0.001 384 $IPDISPLAY
50
51 echo "##### affiche heure #####"
52 date
53
54 echo "##### Start TG 2000 - 1450 #####"
55 sh /$DIRECTORY/fundp/environnement/maxDownlink/tg/maxDownTG.sh $HOUR $DATE udp
       $INTERFACE $DIRECTORY $NODEPL $PROVIDER 1450 0.0058 2000 $IPDISPLAY
56
57 echo "##### affiche heure #####"
58 date
59
60 echo "##### Start TG 2000 - 250 #####"
61 sh /$DIRECTORY/fundp/environnement/maxDownlink/tg/maxDownTG.sh $HOUR $DATE udp
       $INTERFACE $DIRECTORY $NODEPL $PROVIDER 250 0.001 2000 $IPDISPLAY
62
63 echo "##### affiche heure #####"
64 date
65
66 # => only for the WIND network: tg on TCP
67
68 echo "##### Start TG 384 - 1450 #####"
69 sh /$DIRECTORY/fundp/environnement/maxDownlink/tg/maxDownTG.sh $HOUR $DATE tcp
       $INTERFACE $DIRECTORY $NODEPL $PROVIDER 1450 0.030 384 $IPDISPLAY
70
71 echo "##### affiche heure #####"
72 date
73
74 echo "##### Start TG 384 - 48 #####"
75 sh /$DIRECTORY/fundp/environnement/maxDownlink/tg/maxDownTG.sh $HOUR $DATE tcp
       $INTERFACE $DIRECTORY $NODEPL $PROVIDER 48 0.001 384 $IPDISPLAY
76
77 echo "##### affiche heure #####"
78 date
79
80 #echo "##### Start TG 2000 - 1450 #####"
81 #sh /$DIRECTORY/fundp/environnement/maxDownlink/tg/maxDownTG.sh $HOUR $DATE tcp
       $INTERFACE $DIRECTORY $NODEPL $PROVIDER 1450 0.0058 2000 $IPDISPLAY
82
83 echo "##### affiche heure #####"
84 date
85
86 #echo "##### Start TG 2000 - 250 #####"
87 #sh /$DIRECTORY/fundp/environnement/maxDownlink/tg/maxDownTG.sh $HOUR $DATE tcp
```

```
    $INTERFACE $DIRECTORY $NODEPL $PROVIDER 250 0.001 2000 $IPDISPLAY
88
89  echo "##### affiche heure #####"
90  date
91
92  echo "##### Start tracepath #####"
93  sh /$DIRECTORY/fundp/environnement/tracepath/tracepath.sh $HOUR $DATE $DIRECTORY
        $PROVIDER $NODEPL $INTERFACE
94
95  echo "##### affiche heure #####"
96  date
97
98  echo "##### Start Video Streaming - synthetic MQ #####"
99  sh /$DIRECTORY/fundp/videoStreaming/synthetic/videoStreamTG.sh $HOUR $DATE
        $INTERFACE $DIRECTORY $NODEPL $PROVIDER MQ 3 1,5 15,5 2,5 4,5 1 $IPDISPLAY
100
101 echo "##### affiche heure #####"
102 date
103
104 echo "##### Start Video Streaming - synthetic LQ #####"
105 sh /$DIRECTORY/fundp/videoStreaming/synthetic/videoStreamTG.sh $HOUR $DATE
        $INTERFACE $DIRECTORY $NODEPL $PROVIDER LQ 2,75 1 16,25 1,5 3,5 0,5 $IPDISPLAY
106
107 echo "##### affiche heure #####"
108 date
109
110 echo "##### Start Video Streaming - real TCP videoTest -1 #####"
111 sh /$DIRECTORY/fundp/videoStreaming/real/videoStreamingREAL.sh $HOUR $DATE
        $INTERFACE $DIRECTORY $NODEPL $PROVIDER $IPDISPLAY TCP 1
112
113 echo "##### affiche heure #####"
114 date
115
116 echo "##### Start Video Streaming - real TCP videoTest -2 #####"
117 sh /$DIRECTORY/fundp/videoStreaming/real/videoStreamingREAL.sh $HOUR $DATE
        $INTERFACE $DIRECTORY $NODEPL $PROVIDER $IPDISPLAY TCP 2
118
119 echo "##### affiche heure #####"
120 date
121
122 echo "##### Start Video Streaming - real UDP videoTest -1 #####"
123 sh /$DIRECTORY/fundp/videoStreaming/real/videoStreamingREAL.sh $HOUR $DATE
        $INTERFACE $DIRECTORY $NODEPL $PROVIDER $IPDISPLAY UDP 1
124
125 echo "##### affiche heure #####"
126 date
127
128 echo "##### Start Video Streaming - real UDP videoTest -2 #####"
129 sh /$DIRECTORY/fundp/videoStreaming/real/videoStreamingREAL.sh $HOUR $DATE
        $INTERFACE $DIRECTORY $NODEPL $PROVIDER $IPDISPLAY UDP 2
130
131 echo "##### affiche heure #####"
132 date
133
134 echo "##### Start VOIP - file 1 #####"
135 sh /$DIRECTORY/fundp/voip/synthetic/voipTG.sh $HOUR $DATE $INTERFACE $DIRECTORY
        $NODEPL $PROVIDER 1 $IPDISPLAY
136
137 echo "##### affiche heure #####"
138 date
```

```
139
140 echo "##### Start VOIP -- file 2 #####"
141 sh /$DIRECTORY/fundp/voip/synthetic/voipTG.sh $HOUR $DATE $INTERFACE $DIRECTORY
    $NODEPL $PROVIDER 2 $IPDISPLAY
142
143 echo "##### affiche heure #####"
144 date
145
146 echo "##### Start VOIP REAL #####"
147 sh /$DIRECTORY/fundp/voip/real/voipREAL.sh $HOUR $DATE $INTERFACE $DIRECTORY
    $NODEPL $PROVIDER $IPDISPLAY
148
149 echo "##### affiche heure #####"
150 date
151
152 echo "##### stop signal #####"
153 kill -9 $PIDSIG
154 ssh -i ~/.ssh/root_ssh_key.rsa -t root@onelab03.dis.unina.it "killall sh"
155
156 if [ "$INTERFACE" = "ppp0" ]; then
157     if [ "$DIRECTORY" = "root" ]; then
158         echo "Shutdown the UMTS connexion"
159         killall wvdial
160         echo "Shutdown the PCMCIA card"
161         pccardctl eject
162         echo "UMTS closed => end of tests!"
163     else
164         su -c "umts stop"
165     fi
166 else
167     echo "end of tests!"
168 fi
```

E.2 AT-commands

```
1 #!/bin/bash
2 # atCommandsVPOST4.sh
3 # input 1 = HOUR of the experience
4 # input 2 = DATE
5 # input 3 = directory used
6 # input 4 = PROVIDER
7 # input 5 = INTERFACE
8
9 echo "##### Variables configuration #####"
10 HOUR=$1
11 DATE=$2
12 DIRECTORY=$3
13 PROVIDER=$4
14 INTERFACE=$5
15
16 if [ "$DIRECTORY" = "root" ];
17 then
18         if [ "$INTERFACE" = "ppp0" ];
19         then
20                 su -c "route add 138.48.32.100 ppp0"
21         fi
22
23         COUNT=1
```

```
        while (( $COUNT <= 1 ))
        do
                echo "##### AT commands - test $COUNT #####"
                sh /$DIRECTORY/fundp/environnement/atCommands/commands.sh gcom > /
                    $DIRECTORY/fundp/environnement/atCommands/$PROVIDER/$HOUR/
                    atCommandsLB$DATE-$HOUR-$COUNT.txt
                wait
                let COUNT=COUNT+1
        done
else
        if [ "$INTERFACE" = "ppp0" ];
        then
                su -c "umts add 138.48.32.100"
        fi

        COUNT=1
        while (( $COUNT <= 1 ))
        do
                echo "##### AT commands - essai $COUNT #####"
                ssh -i ~/.ssh/root_ssh_key.rsa root@onelab03.dis.unina.it "sh ~/
                    fundp/environnement/atCommands/commands.sh comgt > ~/fundp/
                    environnement/atCommands/$PROVIDER/$HOUR/atCommandsPR$DATE-
                    $HOUR-$COUNT.txt"
                wait
                let COUNT=COUNT+1
        done
fi
```

E.3 Traceroute

```
#!/bin/bash
#tracepathWIND4.sh
# input 1 = Hour
# input 2 = Date
# input 3 = Directory
# input 4 = Provider
# input 5 = Node to reach
# input 6 = Interface

echo "##### Variables configuration #####"
HOUR=$1
DATE=$2
DIRECTORY=$3
PROVIDER=$4
NODETR=$5
INTERFACE=$6

if [ "$DIRECTORY" = "root" ];
then
        PC=$LB
else
        PC=$PR
fi

COUNT=1
while (( $COUNT <= 1 ))
do
        echo "##### tracepath - test $COUNT #####"
```

```
29          STR=`tracepath $NODETR`
30          echo $STR > /$DIRECTORY/fundp/environnement/tracepath/$PROVIDER/$HOUR/
                tracepath$PC$DATE-$HOUR-$COUNT.txt
31          wait
32          echo "Take the first IP address"
33          STR="${STR:3}"
34          # go to the line of the first IP address
35          while [[ "${STR:0:1}" != ":" ]];
36          do
37                  STR="${STR:1}"
38          done
39
40          # go to the beginning of the IP address
41          while [[ "${STR:0:1}" != "(" ]];
42          do
43                  STR="${STR:1}"
44          done
45
46          # put the IP address in IP variable
47          STR="${STR:1}"
48          IP=""
49          while [[ "${STR:0:1}" != ")" ]];
50          do
51                  IP="$IP${STR:0:1}"
52                  STR="${STR:1}"
53          done
54
55          echo "Ping this IP address"
56          ping -I $INTERFACE $IP > /$DIRECTORY/fundp/environnement/tracepath/
                $PROVIDER/$HOUR/ping$PC$DATE-$HOUR-$COUNT.txt &
57          sleep 10
58          killall ping
59          let COUNT=COUNT+1
60  done
```

E.4 Maximal downlink traffic

E.4.1 Script for `tg`

```
1   #!/bin/bash
2   # maxDownTG.sh
3   # input 1 = HOUR of the experience
4   # input 2 = DATE
5   # input 3 = PROTOCOL used
6   # input 4 = network INTERFACE
7   # input 5 = directory used
8   # input 6 = PlanetLab node
9   # input 7 = PROVIDER
10  # input 8 = size of the packet
11  # input 9 = FREQUENCE of packets
12  # input 10 = SPEED
13  # input 11 = Parameter to xterm : display to another host?
14
15  echo "##### Variables configuration #####"
16  HOUR=$1
17  DATE=$2
18  PROTOCOL=$3
19  INTERFACE=$4
```

```
20 DIRECTORY=$5
21 NODEPL=$6
22 PROVIDER=$7
23 PSIZE=$8
24 FREQUENCE=$9
25 SPEED=${10}
26 IPDISPLAY=${11}
27 IPUMTS=`ifconfig $INTERFACE | sed -n 's/.*inet addr:\([^.]*.[^.]*.[^.]*.[^ ]*\).*$
       /\1/p'`
28
29 if [ "$DIRECTORY" = "root" ]; then
30         PC="LB"
31 else
32         PC="PR"
33 fi
34
35 if [ "$DIRECTORY" = "root" ]; then
36         PARAMXTERM=""
37 else
38         PARAMXTERM="-display $IPDISPLAY:0"
39 fi
40
41 echo "##### configuration of TG files #####"
42
43 echo "Creation of MaxDownLinkServer.tg with the IPUMTS: $IPUMTS"
44 echo "# max Downlink server
45 on 0:03 $PROTOCOL $IPUMTS.57001 server
46 at 1.1 wait 75" > /$DIRECTORY/fundp/environnement/maxDownlink/tg/maxDownlinkServer
       .tg
47
48 echo "Creation of MaxDownLinkClient$SPEED-$PSIZE.tg with the IPUMTS: $IPUMTS"
49 echo "# max Downlink client
50 on 0:03 $PROTOCOL $IPUMTS.57001
51 at 1 setup
52 at 2 arrival constant $FREQUENCE length constant $PSIZE
53 time 60" > /$DIRECTORY/fundp/environnement/maxDownlink/tg/maxDownlinkClient$SPEED-
       $PSIZE.tg
54
55 echo "Copy of files on the planetlab node..."
56 scp /$DIRECTORY/fundp/environnement/maxDownlink/tg/maxDownlinkClient*
       uninaonelab_umts@$NODEPL:~/fundp/environnement/maxDownlink/tg/
57
58 COUNT=1
59 while (( $COUNT <= 1 ))
60 do
61         echo "##### maxDownTG - test $COUNT #####"
62
63         echo "run tcpdump..."
64         su -c "tcpdump -i $INTERFACE -s 0 -w /$DIRECTORY/fundp/environnement/
               maxDownlink/tg/$PROVIDER/$HOUR/mdtg$PC-$SPEED-$PSIZE-$DATE-$HOUR-
               $COUNT$PROTOCOL.dump &"
65         sleep 2
66
67         echo "run tcpdump on $NODEPL..."
68         xterm $PARAMXTERM -e ssh -t uninaonelab_umts@$NODEPL "su -c 'tcpdump -i
               eth0 -s 0 -w /home/uninaonelab_umts/fundp/environnement/maxDownlink/tg
               /$PROVIDER/$HOUR/mdtg$PC-PL$SPEED-$PSIZE-$DATE-$HOUR-$COUNT$PROTOCOL.
               dump'" &
69         sleep 4
70
```

```
71          if [ "$PROVIDER" != "WIND" ]; then
72                  echo "!!! open the port on the firewall of the PROVIDER !!!"
73                  ssh uninaonelab_umts@$NODEPL "tg -i ~/fundp/openFirewallS$PROTOCOL
                        .tg"&
74                  PIDFS=$!
75                  sleep 3
76
77                  echo "!!! client to open the firewall !!!"
78                  tgsrc57001reuse -i /$DIRECTORY/fundp/openFirewallC$PROTOCOL.tg &
79                  PIDFC=$!
80                  wait $PIDFC
81                  echo ""
82                  echo "... the client has finished his job..."
83          else
84                  echo "Do not "open" the firewall => WIND"
85          fi
86
87          echo "run tg MaxDownlinkServer..."
88          tgreuse -i /$DIRECTORY/fundp/environnement/maxDownlink/tg/
                maxDownlinkServer.tg -o /$DIRECTORY/fundp/environnement/maxDownlink/tg
                /$PROVIDER/$HOUR/mdServer$PC-$SPEED-$PSIZE-$DATE-$HOUR-$COUNT$PROTOCOL
                .log &
89          PIDS=$!
90          echo "PID of the server:" $PIDS
91          echo ""
92          sleep 2
93
94          if [ "$PROVIDER" != "WIND" ]; then
95                  echo "wait the end of the openFirewall server..."
96                  wait $PIDFS
97          fi
98
99          echo "run the tg MaxDownlinkClient >>> GO"
100         ssh uninaonelab_umts@$NODEPL "tgsrc57002 -i ~/fundp/environnement/
                maxDownlink/tg/maxDownlinkClient$SPEED-$PSIZE.tg -o ~/fundp/
                environnement/maxDownlink/tg/$PROVIDER/$HOUR/mdClient$PC-$SPEED-$PSIZE
                -$DATE-$HOUR-$COUNT$PROTOCOL.log"
101
102         echo "wait the end of the server.... PID: " $PIDS
103         wait $PIDS
104         echo "kill each tcpdump..."
105         su -c "killall tcpdump"
106         ssh -t uninaonelab_umts@$NODEPL "su -c 'killall tcpdump'"
107         reset
108         sleep 4
109         let COUNT=COUNT+1
110 done
```

For 2,000 kb/s the packet size is 1,450 and the frequence is 0.0058 (we also use other figures in order to see if there is a difference: 250 for the size and 0.001 for the frequency). As previously explained, the protocol for this test is UDP.

E.4.2 Script for `wget`

```
1  #!/bin/bash
2  # maxDownWgetVPOST4.sh
3  # input 1 = HOUR of the experience
4  # input 2 = DATE
```

```
# input 3 = network INTERFACE
# input 4 = directory used
# input 5 = PROVIDER
# input 6 = PlanetLab node

echo "##### Variables configuration #####"
HOUR=$1
DATE=$2
INTERFACE=$3
DIRECTORY=$4
PROVIDER=$5
NODEPL=$6

if [ "$DIRECTORY" = "root" ];
then
        PC=$LB
else
        PC=$PR
fi


if [ "$INTERFACE" = "ppp0" ];
then
        echo "add route on ppp0"
        if [ "$DIRECTORY" = "root" ];
        then
                su -c "route add 138.48.32.100 ppp0"
        else
                su -c "umts add 138.48.32.100"
        fi
else
        echo "no route add => we are on eth0"
fi

COUNT=1
while (( $COUNT <= 1 ))
do
        echo "##### maxDownWget - test $COUNT #####"
        su -c "tcpdump -i $INTERFACE -s 0 -w /$DIRECTORY/fundp/environnement/
            maxDownlink/wget/$PROVIDER/$HOUR/mdwget$PC$DATE-$HOUR-$COUNT.dump &"
        sleep 2
        wget www.info.fundp.ac.be/~bevrard/livreReference.pdf -o /$DIRECTORY/fundp
            /environnement/maxDownlink/wget/$PROVIDER/$HOUR/mdwget$PC$DATE-$HOUR-
            $COUNT.txt
        echo "PID of wget:" $PIDS
        wait $PIDS
        su -c "killall tcpdump"
        sleep 15
        let COUNT=COUNT+1
done

echo "##### clean the repository #####"
rm -f /$DIRECTORY/fundp/livre*
```

E.5 Synthetic Video Streaming

```
#!/bin/bash
# maxDownTG.sh
```

```
# input 1 = HOUR of the experience
# input 2 = DATE
# input 3 = network INTERFACE
# input 4 = directory used
# input 5 = PlanetLab node
# input 6 = PROVIDER
# input 7 = quality
# input 8,9,10,11,12,13 = parameters for the gamma distribution
# input 14 = Parameter to xterm : display to another host?


echo "##### Variables configuration #####"
HOUR=$1
DATE=$2
INTERFACE=$3
DIRECTORY=$4
NODEPL=$5
PROVIDER=$6
QUALITY=$7
ALPHA1=$8
ALPHA2=${9}
ALPHA3=${10}
LAMB1=${11}
LAMB2=${12}
LAMB3=${13}
IPDISPLAY=${14}
IPUMTS=`ifconfig $INTERFACE | sed -n 's/.*inet addr:\([^.]*.[^.]*.[^.]*.[^ ]*\).*$
    /\1/p'`

if [ "$DIRECTORY" = "root" ]; then
        PC="LB"
else
        PC="PR"
fi

if [ "$DIRECTORY" = "root" ]; then
        PARAMXTERM=""
else
        PARAMXTERM="-display $IPDISPLAY:0"
fi

echo "##### configuration of TG files #####"

echo "Creation of videoStreamServer.tg with the IPUMTS: $IPUMTS"
echo "# max Downlink server
on 0:03 udp $IPUMTS.57001 server
at 1.1 wait 130" > /$DIRECTORY/fundp/videoStreaming/synthetic/videoStreamServer.tg

echo "Creation of videStreamClient$QUALITY.tg with the IPUMTS: $IPUMTS"
/$DIRECTORY/fundp/videoStreaming/synthetic/streamReal $IPUMTS 57001 3000 $ALPHA1
    $ALPHA2 $ALPHA3 $LAMB1 $LAMB2 $LAMB3 > /$DIRECTORY/fundp/videoStreaming/
    synthetic/videoStreamClient$QUALITY.tg

echo "Copy of files on the planetlab node..."
scp /$DIRECTORY/fundp/videoStreaming/synthetic/videoStreamClient*
    uninaonelab_umts@$NODEPL:~/fundp/videoStreaming/synthetic/

COUNT=1
while (( $COUNT <= 1 ))
do
```

```
59          echo "##### videoStream - test $COUNT #####"
60
61          echo "run tcpdump..."
62          su -c "tcpdump -i $INTERFACE -s 0 -w /$DIRECTORY/fundp/videoStreaming/
                synthetic/$PROVIDER/$HOUR/videoStream$PC-$QUALITY-$DATE-$HOUR-$COUNT.
                dump &"
63          sleep 2
64
65          echo "run tcpdump on $NODEPL..."
66          xterm $PARAMXTERM -e ssh -t uninaonelab_umts@$NODEPL "su -c 'tcpdump -i
                eth0 -s 0 -w /home/uninaonelab_umts/fundp/videoStreaming/synthetic/
                $PROVIDER/$HOUR/videoStream$PC-PL$QUALITY-$DATE-$HOUR-$COUNT.dump'" &
67          sleep 2
68
69          if [ "$PROVIDER" != "WIND" ]; then
70                  echo "!!! open the port on the firewall of the PROVIDER !!!"
71                  ssh uninaonelab_umts@$NODEPL "tg -i ~/fundp/openFirewallSudp.tg"&
72                  PIDFS=$!
73                  sleep 1
74
75                  echo "!!! client to open the firewall !!!"
76                  tgsrc57001reuse -i /$DIRECTORY/fundp/openFirewallCudp.tg &
77                  PIDFC=$!
78                  wait $PIDFC
79                  echo ""
80                  echo "... the client has finished his job..."
81          else
82                  echo "Do not "open" the firewall => WIND"
83          fi
84
85          echo "run tg videoStreamServer..."
86          tgreuse -i /$DIRECTORY/fundp/videoStreaming/synthetic/videoStreamServer.tg
                -o /$DIRECTORY/fundp/videoStreaming/synthetic/$PROVIDER/$HOUR/
                videoStreamServer$PC-$QUALITY-$DATE-$HOUR-$COUNT.log &
87          PIDS=$!
88          echo "PID of the server:" $PIDS
89          echo ""
90
91          if [ "$PROVIDER" != "WIND" ]; then
92                  echo "wait the end of the openFirewall server..."
93                  wait $PIDFS
94          fi
95
96          echo "run the tg videoStreamClient$QUALITY >>> GO"
97          ssh uninaonelab_umts@$NODEPL "tgsrc57002 -i ~/fundp/videoStreaming/
                synthetic/videoStreamClient$QUALITY.tg -o ~/fundp/videoStreaming/
                synthetic/$PROVIDER/$HOUR/videoStreamClient$PC-$QUALITY-$DATE-$HOUR-
                $COUNT.log"
98          echo "wait the end of the server.... PID: " $PIDS
99          wait $PIDS
100         echo "kill each tcpdump..."
101         su -c "killall tcpdump"
102         ssh -t uninaonelab_umts@$NODEPL "su -c 'killall tcpdump'"
103         reset
104         sleep 4
105         let COUNT=COUNT+1
106 done
```

E.6 Synthetic VoIP

```
#!/bin/bash
# maxDownTG.sh
# input 1 = HOUR of the experience
# input 2 = DATE
# input 3 = network INTERFACE
# input 4 = directory used
# input 5 = PlanetLab node
# input 6 = PROVIDER
# input 7 = file to simulate voip
# input 8 = Parameter to xterm : display to another host?

echo "##### Variables configuration #####"
HOUR=$1
DATE=$2
INTERFACE=$3
DIRECTORY=$4
NODEPL=$5
PROVIDER=$6
FILE=$7
IPDISPLAY=$8
IPUMTS=`ifconfig $INTERFACE | sed -n 's/.*inet addr:\([^.]*.[^.]*.[^.]*.[^ ]*\).*$
    /\1/p'`

if [ "$DIRECTORY" = "root" ];
then
        PC="LB"
else
        PC="PR"
fi

if [ "$DIRECTORY" = "root" ]; then
        PARAMXTERM=""
else
        PARAMXTERM="-display $IPDISPLAY:0"
fi

echo "##### configuration of TG files #####"

echo "Creation of voipServer.tg with the IPUMTS: $IPUMTS"
echo "# max Downlink server
on 0:03 udp $IPUMTS.57001 server
at 1.1 wait 100" > /$DIRECTORY/fundp/voip/synthetic/voipServer.tg

echo "Creation of voipClient with the IPUMTS: $IPUMTS"
sh /$DIRECTORY/fundp/voip/synthetic/createFilesVOIP $DIRECTORY $IPUMTS

echo "Copy of files on the planetlab node..."
scp /$DIRECTORY/fundp/voip/synthetic/voipClient* uninaonelab_umts@$NODEPL:~/fundp/
    voip/synthetic/

COUNT=1
while (( $COUNT <= 1 ))
do
        echo "##### voipTG - test $COUNT #####"

        echo "run tcpdump..."
```

```
55          su -c "tcpdump -i $INTERFACE -s 0 -w /$DIRECTORY/fundp/voip/synthetic/
                $PROVIDER/$HOUR/voipTG$PC-$FILE-$DATE-$HOUR-$COUNT.dump &"
56          sleep 2
57
58          echo "run tcpdump on $NODEPL..."
59          xterm $PARAMXTERM -e ssh -t uninaonelab_umts@$NODEPL "su -c 'tcpdump -i
                eth0 -s 0 -w /home/uninaonelab_umts/fundp/voip/synthetic/$PROVIDER/
                $HOUR/voipTG$PC-PL$FILE-$DATE-$HOUR-$COUNT.dump'"&
60          sleep 2
61
62          if [ "$PROVIDER" != "WIND" ]; then
63                  echo "!!! open the port on the firewall of the PROVIDER !!!"
64                  ssh uninaonelab_umts@$NODEPL "tg -i ~/fundp/openFirewallSudp.tg"&
65                  PIDFS=$!
66                  sleep 1
67
68                  echo "!!! client to open the firewall !!!"
69                  tgsrc57001reuse -i /$DIRECTORY/fundp/openFirewallCudp.tg &
70                  PIDFC=$!
71                  wait $PIDFC
72                  echo ""
73                  echo "... the client has finished his job..."
74          else
75                  echo "Do not "open" the firewall => WIND"
76          fi
77
78          echo "run tg voipServer..."
79          tgreuse -i /$DIRECTORY/fundp/voip/synthetic/voipServer.tg -o /$DIRECTORY/
                fundp/voip/synthetic/$PROVIDER/$HOUR/voipServer$PC-$FILE-$DATE-$HOUR-
                $COUNT.log &
80          PIDS=$!
81          echo "PID of the server:" $PIDS
82          echo ""
83
84          if [ "$PROVIDER" != "WIND" ]; then
85                  echo "wait the end of the openFirewall server..."
86                  wait $PIDFS
87          fi
88
89          echo "run the tg voipClient$FILE >>> GO"
90          ssh uninaonelab_umts@$NODEPL "tgsrc57002 -i ~/fundp/voip/synthetic/
                voipClient$FILE.tg -o ~/fundp/voip/synthetic/$PROVIDER/$HOUR/
                voipClient$PC-$FILE-$DATE-$HOUR-$COUNT.log"
91          echo "wait the end of the server.... PID: " $PIDS
92          wait $PIDS
93          echo "kill each tcpdump..."
94          su -c "killall tcpdump"
95          ssh -t uninaonelab_umts@$NODEPL "su -c 'killall tcpdump'"
96          reset
97          sleep 4
98          let COUNT=COUNT+1
99  done
```

E.7 Video Streaming Real

```
1   #!/bin/bash
2   # videoStreamingREAL.sh
3   #
```

```
# input 1 = HOUR of the experience
# input 2 = DATE
# input 3 = network INTERFACE
# input 4 = directory used
# input 5 = PlanetLab node
# input 6 = PROVIDER
# input 7 = Parameter to xterm : display to another host?
# input 8 = PROTOCOL
# input 9 = VIDEO id

echo "##### Variables configuration #####"
HOUR=$1
DATE=$2
INTERFACE=$3
DIRECTORY=$4
NODEPL=$5
PROVIDER=$6
IPDISPLAY=$7
PROTOCOL=$8
VIDEO=$9

IPUMTS=`ifconfig $INTERFACE | sed -n 's/.*inet addr:\([^.]*.[^.]*.[^.]*.[^ ]*\).*$
    /\1/p'`

if [ "$DIRECTORY" = "root" ];
then
        PC="LB"
else
        PC="PR"
fi

if [ "$DIRECTORY" = "root" ]; then
        PARAMXTERM=""
else
        PARAMXTERM="-display $IPDISPLAY:0"
fi

if [ "$PROTOCOL" = "TCP" ];
then
        PROTPARAM="-t"
else
        PROTPARAM="-v"
fi

if [ "$VIDEO" = "1" ];
then
        FRAME="25"
else
        FRAME="30"
fi

COUNT=1
while (( $COUNT <= 1 ))
do
        echo "##### videoStreamingREAL - test $COUNT #####"

        echo "run tcpdump..."
        su -c "tcpdump -i $INTERFACE -s 0 -w /$DIRECTORY/fundp/videoStreaming/real
            /$PROVIDER/$HOUR/videoStreamingREAL$PC-$VIDEO$PROTOCOL-$DATE-$HOUR-
            $COUNT.dump &"
```

```
61          sleep 4
62
63          echo "run tcpdump on $NODEPL..."
64          xterm $PARAMXTERM -e ssh -t uninaonelab_umts@$NODEPL "su -c 'tcpdump -i
                eth0 -s 0 -w /home/uninaonelab_umts/fundp/videoStreaming/real/
                $PROVIDER/$HOUR/videoStreamingREAL$PC-PL-$VIDEO$PROTOCOL-$DATE-$HOUR-
                $COUNT.dump'"&
65          sleep 4
66
67          echo "##### Run openRTSP (on $PC) - Down on $PROTOCOL - videoTest-$VIDEO.
                mp4 #####"
68          openRTSP -f $FRAME -w 320 -h 240 -4 -n -Q $PROTPARAM -v rtsp://onelab09.
                inria.fr:8554/videoTest-$VIDEO.mp4 > /$DIRECTORY/fundp/videoStreaming/
                real/$PROVIDER/$HOUR/videoStreamingREAL$PC-$VIDEO$PROTOCOL-$DATE-$HOUR
                -$COUNT.mp4 2> /$DIRECTORY/fundp/videoStreaming/real/$PROVIDER/$HOUR/
                videoStreamingREAL$PC-$VIDEO$PROTOCOL-$DATE-$HOUR-$COUNT.txt
69          let COUNT=COUNT+1
70
71          echo "kill each tcpdump..."
72          su -c "killall tcpdump"
73          ssh -t uninaonelab_umts@$NODEPL "su -c 'killall tcpdump'"
74          reset
75          sleep 4
76  done
```

E.8 VoIP Real

```
1   #!/bin/bash
2   # voipREAL.sh
3   #
4   # input 1 = HOUR of the experience
5   # input 2 = DATE
6   # input 3 = network INTERFACE
7   # input 4 = directory used
8   # input 5 = PlanetLab node
9   # input 6 = PROVIDER
10  # input 7 = Parameter to xterm : display to another host?
11
12  echo "##### Variables configuration #####"
13  HOUR=$1
14  DATE=$2
15  INTERFACE=$3
16  DIRECTORY=$4
17  NODEPL=$5
18  PROVIDER=$6
19  IPDISPLAY=$7
20
21  IPUMTS=`ifconfig $INTERFACE | sed -n 's/.*inet addr:\([^.]*.[^.]*.[^.]*.[^ ]*\).*$
        /\1/p'`
22
23  if [ "$DIRECTORY" = "root" ];
24  then
25          PC="LB"
26  else
27          PC="PR"
28  fi
29
30  if [ "$DIRECTORY" = "root" ]; then
```

```
31          PARAMXTERM=""
32  else
33          PARAMXTERM="-display $IPDISPLAY:0"
34  fi
35
36  COUNT=1
37  while (( $COUNT <= 1 ))
38  do
39          echo "##### voipREAL - test $COUNT #####"
40
41          echo "run tcpdump..."
42          su -c "tcpdump -i $INTERFACE -s 0 -w /$DIRECTORY/fundp/voip/real/$PROVIDER
                /$HOUR/voipREAL$PC-$DATE-$HOUR-$COUNT.dump &"
43          sleep 4
44
45          echo "run tcpdump on $NODEPL..."
46          xterm $PARAMXTERM -e ssh -t uninaonelab_umts@$NODEPL "su -c 'tcpdump -i
                eth0 -s 0 -w /home/uninaonelab_umts/fundp/voip/real/$PROVIDER/$HOUR/
                voipREAL$PC-PL-$DATE-$HOUR-$COUNT.dump'"&
47          sleep 4
48
49          echo "##### Run SIPp server (on $NODEPL) #####"
50          xterm $PARAMXTERM -e ssh -t uninaonelab_umts@$NODEPL "su -c 'sipp -sf /
                home/uninaonelab_umts/fundp/voip/real/uas_for_real_voiptest -2minutes.
                xml -i 138.96.250.149 -p 9002 -mi 138.96.250.149 -mp 9902 -m 1 -
                trace_msg -trace_shortmsg -trace_screen -trace_err -trace_stat -
                trace_counts -trace_rtt -trace_logs '"&
51          PIDSIPS=$!
52
53          # wait to be sure that the server is running...
54          sleep 3
55
56          echo "##### Run SIPp client (on $PC) #####"
57          su -c "sipp 138.96.250.149:9002 -i $IPUMTS -p 9001 -mi $IPUMTS -mp 9901 -
                sf /$DIRECTORY/fundp/voip/real/uac_for_real_voiptest -2minutes-$PC.xml
                -l 1 -r 1 -m 1 -trace_msg -trace_screen -trace_err -trace_stat -
                trace_rtt -trace_logs"
58          PIDSIPC=$!
59
60          echo ""
61          echo "wait the end of the client..."
62          wait $PIDSIPC
63
64          echo "wait the end of the server..."
65          wait $PIDSIPS
66
67          echo "##### Move log files on $PC #####"
68          mv /$DIRECTORY/fundp/voip/real/*_.csv /$DIRECTORY/fundp/voip/real/
                $PROVIDER/$HOUR/voipREALClient$PC-$DATE-$HOUR-$COUNT-stats.csv
69          mv /$DIRECTORY/fundp/voip/real/*logs.log /$DIRECTORY/fundp/voip/real/
                $PROVIDER/$HOUR/voipREALClient$PC-$DATE-$HOUR-$COUNT-logs.log
70          mv /$DIRECTORY/fundp/voip/real/*_messages.log /$DIRECTORY/fundp/voip/real/
                $PROVIDER/$HOUR/voipREALClient$PC-$DATE-$HOUR-$COUNT-messages.log
71          mv /$DIRECTORY/fundp/voip/real/*rtt.csv /$DIRECTORY/fundp/voip/real/
                $PROVIDER/$HOUR/voipREALClient$PC-$DATE-$HOUR-$COUNT-rtt.csv
72          mv /$DIRECTORY/fundp/voip/real/*screen.log /$DIRECTORY/fundp/voip/real/
                $PROVIDER/$HOUR/voipREALClient$PC-$DATE-$HOUR-$COUNT-screen.log
73          mv /$DIRECTORY/fundp/voip/real/*errors.log /$DIRECTORY/fundp/voip/real/
                $PROVIDER/$HOUR/voipREALClient$PC-$DATE-$HOUR-$COUNT-errors.log
74
```

```
75          echo "##### Move log files on $NODEPL #####"
76          ssh -t uninaonelab_umts@$NODEPL "su -c 'mv /home/uninaonelab_umts/fundp/
                voip/real/*counts.csv /home/uninaonelab_umts/fundp/voip/real/$PROVIDER
                /$HOUR/voipREALServer$PC-PL-$DATE-$HOUR-$COUNT-counts.csv '"
77          ssh -t uninaonelab_umts@$NODEPL "su -c 'mv /home/uninaonelab_umts/fundp/
                voip/real/*_.csv /home/uninaonelab_umts/fundp/voip/real/$PROVIDER/
                $HOUR/voipREALServer$PC-PL-$DATE-$HOUR-$COUNT-stats.csv '"
78          ssh -t uninaonelab_umts@$NODEPL "su -c 'mv /home/uninaonelab_umts/fundp/
                voip/real/*logs.log /home/uninaonelab_umts/fundp/voip/real/$PROVIDER/
                $HOUR/voipREALServer$PC-PL-$DATE-$HOUR-$COUNT-logs.log '"
79          ssh -t uninaonelab_umts@$NODEPL "su -c 'mv /home/uninaonelab_umts/fundp/
                voip/real/*_messages.log /home/uninaonelab_umts/fundp/voip/real/
                $PROVIDER/$HOUR/voipREALServer$PC-PL-$DATE-$HOUR-$COUNT-messages.log '"
80          ssh -t uninaonelab_umts@$NODEPL "su -c 'mv /home/uninaonelab_umts/fundp/
                voip/real/*rtt.csv /home/uninaonelab_umts/fundp/voip/real/$PROVIDER/
                $HOUR/voipREALServer$PC-PL-$DATE-$HOUR-$COUNT-rtt.csv '"
81          ssh -t uninaonelab_umts@$NODEPL "su -c 'mv /home/uninaonelab_umts/fundp/
                voip/real/*screen.log /home/uninaonelab_umts/fundp/voip/real/$PROVIDER
                /$HOUR/voipREALServer$PC-PL-$DATE-$HOUR-$COUNT-screen.log '"
82          ssh -t uninaonelab_umts@$NODEPL "su -c 'mv /home/uninaonelab_umts/fundp/
                voip/real/*shortmessages.log /home/uninaonelab_umts/fundp/voip/real/
                $PROVIDER/$HOUR/voipREALServer$PC-PL-$DATE-$HOUR-$COUNT-shortmessages.
                log '"
83          ssh -t uninaonelab_umts@$NODEPL "su -c 'mv /home/uninaonelab_umts/fundp/
                voip/real/*errors.log /home/uninaonelab_umts/fundp/voip/real/$PROVIDER
                /$HOUR/voipREALServer$PC-PL-$DATE-$HOUR-$COUNT-errors.log '"
84
85          echo "kill each tcpdump..."
86          su -c "killall tcpdump"
87          ssh -t uninaonelab_umts@$NODEPL "su -c 'killall tcpdump'"
88          reset
89          sleep 4
90          let COUNT=COUNT+1
91  done
```

E.9 SIPp XML Scenarios

The three following SIPp XML scenarios files can be found at [74].

SIPp XML scenario file on the client side (Linux Box)

```
1   <?xml version="1.0" encoding="ISO-8859-1" ?>
2   <!DOCTYPE scenario SYSTEM "sipp.dtd">
3
4   <!--                                                                    -->
5   <!--                  Sipp 'uac' scenario with pcap (rtp) play          -->
6   <!--                                                                    -->
7
8   <scenario name="UAC with media">
9     <send retrans="500">
10      <![CDATA[
11
12        INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
13        Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
14        From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
15        To: sut <sip:[service]@[remote_ip]:[remote_port]>
16        Call-ID: [call_id]
17        CSeq: 1 INVITE
```

```
18        Contact: sip:sipp@[local_ip]:[local_port]
19        Max-Forwards: 70
20        Subject: Performance Test
21        Content-Type: application/sdp
22        Content-Length: [len]
23
24        v=0
25        o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
26        s=-
27        c=IN IP[local_ip_type] [local_ip]
28        t=0 0
29        m=audio [auto_media_port] RTP/AVP 3 101
30        a=rtpmap:3 gsm/8000/1
31        a=rtpmap:101 telephone-event/8000
32        a=fmtp:101 0-15,32-49
33
34      ]]>
35    </send>
36
37    <recv response="100" optional="true">
38    </recv>
39
40    <recv response="180" optional="true">
41    </recv>
42
43    <recv response="200" rtd="true" crlf="true">
44    </recv>
45
46    <send>
47      <![CDATA[
48
49        ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
50        Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
51        From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
52        To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
53        Call-ID: [call_id]
54        CSeq: 1 ACK
55        Contact: sip:sipp@[local_ip]:[local_port]
56        Max-Forwards: 70
57        Subject: Performance Test
58        Content-Length: 0
59
60      ]]>
61    </send>
62
63    <!-- Play a pre-recorded PCAP file (RTP stream)                    -->
64    <nop>
65      <action>
66        <exec play_pcap_audio="/root/fundp/voip/real/pcap_GSM_for_real_voiptest-
              openingFirewall-10sec.pcap"/>
67      </action>
68    </nop>
69
70    <!-- Pause 11 seconds, which is approximately the duration of the PCAP file
                          -->
71    <pause milliseconds="11000"/>
72
73    <!-- Pause 130 seconds, which is approximately the duration of the PCAP file
                         -->
74    <pause milliseconds="130000"/>
```

```
75
76    <send retrans="500">
77      <![CDATA[
78
79        BYE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
80        Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
81        From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
82        To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
83        Call-ID: [call_id]
84        CSeq: 2 BYE
85        Contact: sip:sipp@[local_ip]:[local_port]
86        Max-Forwards: 70
87        Subject: Performance Test
88        Content-Length: 0
89
90      ]]>
91    </send>
92
93    <recv response="200" crlf="true">
94    </recv>
95
96    <!-- definition of the response time repartition table (unit is ms)  -->
97    <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200, 250, 300,
          350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950, 1000"/>
98
99    <!-- definition of the call length repartition table (unit is ms)  -->
100   <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000, 60000,
          100000, 120000"/>
101
102 </scenario>
```

SIPp XML scenario file on the client side (PrivateLab Node)

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2  <!DOCTYPE scenario SYSTEM "sipp.dtd">
3
4  <!--                                                                  -->
5  <!--                  Sipp 'uac' scenario with pcap (rtp) play        -->
6  <!--                                                                  -->
7
8  <scenario name="UAC with media">
9    <send retrans="500">
10     <![CDATA[
11
12       INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
13       Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
14       From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
15       To: sut <sip:[service]@[remote_ip]:[remote_port]>
16       Call-ID: [call_id]
17       CSeq: 1 INVITE
18       Contact: sip:sipp@[local_ip]:[local_port]
19       Max-Forwards: 70
20       Subject: Performance Test
21       Content-Type: application/sdp
22       Content-Length: [len]
23
24       v=0
25       o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
26       s=-
27       c=IN IP[local_ip_type] [local_ip]
```

```
28          t=0 0
29          m=audio [auto_media_port] RTP/AVP 3 101
30          a=rtpmap:3 gsm/8000/1
31          a=rtpmap:101 telephone-event/8000
32          a=fmtp:101 0-15,32-49
33
34        ]]>
35      </send>
36
37      <recv response="100" optional="true">
38      </recv>
39
40      <recv response="180" optional="true">
41      </recv>
42
43      <recv response="200" rtd="true" crlf="true">
44      </recv>
45
46      <send>
47        <![CDATA[
48
49          ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
50          Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
51          From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
52          To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
53          Call-ID: [call_id]
54          CSeq: 1 ACK
55          Contact: sip:sipp@[local_ip]:[local_port]
56          Max-Forwards: 70
57          Subject: Performance Test
58          Content-Length: 0
59
60        ]]>
61      </send>
62
63      <!-- Play a pre-recorded PCAP file (RTP stream)                    -->
64      <nop>
65        <action>
66          <exec play_pcap_audio="/home/unina_umts/fundp/voip/real/
                pcap_GSM_for_real_voiptest-openingFirewall-10sec.pcap"/>
67        </action>
68      </nop>
69
70      <!-- Pause 11 seconds, which is approximately the duration of the PCAP file
                                -->
71      <pause milliseconds="11000"/>
72
73      <!-- Pause 130 seconds, which is approximately the duration of the PCAP file
                               -->
74      <pause milliseconds="130000"/>
75
76      <send retrans="500">
77        <![CDATA[
78
79          BYE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
80          Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
81          From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
82          To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
83          Call-ID: [call_id]
84          CSeq: 2 BYE
```

```
85          Contact: sip:sipp@[local_ip]:[local_port]
86          Max-Forwards: 70
87          Subject: Performance Test
88          Content-Length: 0
89
90        ]]>
91    </send>
92
93    <recv response="200" crlf="true">
94    </recv>
95
96    <!-- definition of the response time repartition table (unit is ms)  -->
97    <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200, 250, 300,
          350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950, 1000"/>
98
99    <!-- definition of the call length repartition table (unit is ms)    -->
100   <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000, 60000,
          100000, 120000"/>
101
102 </scenario>
```

SIPp XML scenario file on the server side (PlanetLab Node)

```
1   <?xml version="1.0" encoding="ISO-8859-1" ?>
2   <!DOCTYPE scenario SYSTEM "sipp.dtd">
3
4   <!--                                                                -->
5   <!--                 Sipp 'uas' scenario with pcap (rtp) play       -->
6   <!--                                                                -->
7
8   <scenario name="Basic UAS responder">
9     <recv request="INVITE" crlf="true">
10    </recv>
11
12    <send>
13      <![CDATA[
14
15        SIP/2.0 180 Ringing
16        [last_Via:]
17        [last_From:]
18        [last_To:];tag=[call_number]
19        [last_Call-ID:]
20        [last_CSeq:]
21        Contact: <sip:[local_ip]:[local_port];transport=[transport]>
22        Content-Length: 0
23
24      ]]>
25    </send>
26
27    <send retrans="500">
28      <![CDATA[
29
30        SIP/2.0 200 OK
31        [last_Via:]
32        [last_From:]
33        [last_To:];tag=[call_number]
34        [last_Call-ID:]
35        [last_CSeq:]
36        Contact: <sip:[local_ip]:[local_port];transport=[transport]>
37        Content-Type: application/sdp
```

```
38        Content-Length: [len]
39
40        v=0
41        o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
42        s=-
43        c=IN IP[media_ip_type] [media_ip]
44        t=0 0
45        m=audio [auto_media_port] RTP/AVP 3 101
46        a=rtpmap:3 gsm/8000/1
47        a=rtpmap:101 telephone-event/8000
48        a=fmtp:101 0-15,32-49
49
50      ]]>
51    </send>
52
53     <!--      optional="true"  -->
54    <recv request="ACK"
55          rtd="true"
56          crlf="true">
57    </recv>
58
59    <!-- Pause 11 seconds during receiving the opening firewall stream
                                  -->
60    <pause milliseconds="11000"/>
61
62    <!-- Play a pre-recorded PCAP file (RTP stream)                    -->
63    <nop>
64      <action>
65        <exec play_pcap_audio="/home/uninaonelab_umts/fundp/voip/real/
              pcap_GSM_for_real_voiptest-2minutes.pcap"/>
66      </action>
67    </nop>
68
69    <!-- Pause 128 seconds, which is approximately the duration of the PCAP file
                             -->
70    <pause milliseconds="128000"/>
71
72    <recv request="BYE">
73    </recv>
74
75    <send>
76      <![CDATA[
77
78        SIP/2.0 200 OK
79        [last_Via:]
80        [last_From:]
81        [last_To:]
82        [last_Call-ID:]
83        [last_CSeq:]
84        Contact: <sip:[local_ip]:[local_port];transport=[transport]>
85        Content-Length: 0
86
87      ]]>
88    </send>
89
90    <!-- Keep the call open for a while in case the 200 is lost to be     -->
91    <!-- able to retransmit it if we receive the BYE again.               -->
92    <pause milliseconds="4000"/>
93
94    <!-- definition of the response time repartition table (unit is ms)   -->
```

```
95    <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200, 250, 300,
          350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950, 1000"/>
96
97    <!-- definition of the call length repartition table (unit is ms)     -->
98    <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000, 60000,
          100000, 120000"/>
99
100 </scenario>
```

Appendix F

Scripts used to compute the results

F.1 Jitter

The following scripts can be found at [74].

F.1.1 Main scripts

Script to compute the jitter for synthetic traffic (using F.1.2).

```
#!/bin/bash
# test.sh

# input 1 = TG server file
# input 2 = TG client file
# input 3 = output file name

echo "##### Variables configuration #####"
RECEIVE=$1
SEND=$2
OUTPUT=$3

echo "##### dcat #####"
# take a binary and return a text file.
TEMP=${RECEIVE/.log}
dcat $RECEIVE > $TEMP".txt"
RECEIVE=$TEMP".txt"
TEMP=${SEND/.log}
dcat $SEND > $TEMP".txt"
SEND=$TEMP".txt"

echo "##### server file #####"
# take the file of the TG server and return only values of times and ID
# ordered on the ID.
# the last line is removed because it doen't refer a packet.
awk 'NR == 17 , NR == EOF {print $4 " " $1 }' $RECEIVE > receive.txt
head -n -1 receive.txt  > sort.txt
sort -n +0 -1 sort.txt > server.txt


echo "##### client file #####"
# See server file.
awk 'NR == 17 , NR == EOF {print $4 " " $1}' $SEND > send.txt
```

```
34 head -n -1 send.txt  > client.txt
35
36 echo "##### compute jitter #####"
37 computeJitterGeneral server.txt client.txt $OUTPUT
38
39
40 #echo "##### clean #####"
41 rm $RECEIVE
42 rm $SEND
43 rm receive.txt
44 rm send.txt
45 rm sort.txt
46 rm server.txt
47 rm client.txt
```

Script to compute the jitter for real traffic (using F.1.2).

```
1 #!/bin/bash
2 # test.sh
3
4 # input 1 = recoi file
5 # input 2 = envoi file
6 # input 3 = output file name
7
8 echo "##### sort input files #####"
9 sort -n +0 -1 $1 > $1.sort
10 sort -n +0 -1 $2 > $2.sort
11
12 echo "##### compute jitter #####"
13 computeJitterGeneral $1.sort $2.sort $3
14
15 rm $1.sort
16 rm $2.sort
```

F.1.2 computeJitterGeneral.c

This program written in the C language computes the jitter as defined in Section 9.1.

```
1 /* ############################################################
2                       ComputeJitter.c
3 ############################################################*/
4 #include "stdio.h"
5 #include "math.h"
6
7 #define nbMaxPackets 15000
8
9 int main(int argc, char *argv[]){
10   FILE * fileEnvoi;
11   FILE * fileRecoi;
12   FILE * jitter;
13   float envoi[2][nbMaxPackets];
14   float recoi[2][nbMaxPackets];
15   int packetsRecoi=0;
16   int packetsEnvoi=0;
17   char *output;
18
19   // Output file
20   output=argv[3];
21
```

```
22    // Loading of files
23    float id=0;
24    float time=0;
25
26    if ((fileRecoi = fopen(argv[1],"r")) == NULL)
27      printf("Impossible to open the file\n");
28    else{
29      while(fscanf(fileRecoi,"%f %f",&id,&time) != EOF){
30        recoi[0][packetsRecoi]=id;
31        recoi[1][packetsRecoi]=time;
32        packetsRecoi++;
33      }
34    }
35    fclose(fileRecoi);
36
37    if ((fileEnvoi = fopen(argv[2],"r")) == NULL)
38      printf("Impossible to open the file\n");
39    else{
40      while(fscanf(fileEnvoi,"%f %f",&id,&time) != EOF){
41        envoi[0][packetsEnvoi]=id;
42        envoi[1][packetsEnvoi]=time;
43        packetsEnvoi++;
44      }
45    }
46    fclose(fileEnvoi);
47
48    // Check packets lost
49          float packetsR=packetsRecoi;
50          float packetsE=packetsEnvoi;
51    float lost = (packetsR*100)/packetsE;
52    printf("Pourcentage of packets lost:%f\n",(100-lost));
53    printf("Packets started: %d - Packets arrived: %d\n",packetsEnvoi,packetsRecoi);
54
55
56  //elagage des tableaux
57    float OKenvoi[2][nbMaxPackets];
58    float OKrecoi[2][nbMaxPackets];
59
60          int t=0;
61          int i=0;
62          while(i<=(packetsEnvoi-1)){
63                  int boolean=0;
64                  int j=0;
65                  while((boolean==0) && (j<=(packetsRecoi-1))){
66                          if(envoi[0][i]==recoi[0][j])
67                                  {boolean=1;}
68                  j++;
69                  }
70
71                  if(boolean==1){
72                          OKenvoi[0][t]=envoi[0][i];
73                          OKenvoi[1][t]=envoi[1][i];
74                          OKrecoi[0][t]=recoi[0][j-1];
75                          OKrecoi[1][t]=recoi[1][j-1];
76                          t++;}
77          i++;
78          }
79
80    // Compute Jitter
81    if ((jitter = fopen(output,"w")) == NULL)
```

```
82          printf("Impossible to open the file\n");
83    else{
84          float delay_n;
85          float delay_n_1;
86          float jit=0;
87          int u=1;
88          while(u<=(t-1)){
89                  delay_n=OKrecoi[1][u]-OKenvoi[1][u];
90                  delay_n_1=OKrecoi[1][u-1]-OKenvoi[1][u-1];
91                  fprintf(jitter,"%d %f\n",u,fabs(delay_n-delay_n_1));
92                  jit=jit+fabs(delay_n-delay_n_1);
93                  u++;
94          }
95          printf("Average Jitter (in seconds):%f\n",(jit/u));
96          fclose(jitter);
97    }
98 }
```

Appendix G

Part of results of tests and experiments

G.1 Static Approach - Tests

G.1.1 Test 3 - Cell id

	Cell id - [hexadecimal form]		
System	VPOST	VPRE	WIND
PrivateLab Node	026F	026F	0CB8
Linux Box	026F	026F	0CB8

G.1.2 Test 7 - Maximal downlink traffic

`wget`

	`wget` - [Mbps]			
System	ETH	VPOST	VPRE	WIND
PrivateLab Node	10.45	1.371	1.254	1.412
Linux Box	58.391	1.447	1.371	1.573

`tg`

	`tg` (2 Mbps - packet size : 1,450 Bytes) - [Mbps]			
System	ETH	VPOST	VPRE	WIND
PrivateLab Node	2.065	1.390	1.403	1.598
Linux Box	2.064	1.460	1.443	1.614

	`tg` (2 Mbps - packet size : 250 Bytes) - [Mbps]			
System	ETH	VPOST	VPRE	WIND
PrivateLab Node	2.354	1.353	1.455	1.572
Linux Box	2.359	1.539	1.506	1.684

	`tg` Packets lost (2 Mbits/s - packet size : 1,450 Bytes) - [%]			
System	ETH	VPOST	VPRE	WIND
PrivateLab Node	0	31.374	30.785	23.237
Linux Box	0	27.936	28.736	19.698

	`tg` Packets lost (2 Mbits/s, packet size : 250 Bytes) - [%]			
System	ETH	VPOST	VPRE	WIND
PrivateLab Node	0.625	42.937	38.614	34.545
Linux Box	0.403	35.049	36.492	31.599

	`tg` Average Jitter (2 Mbits/s - packet size : 1,450 Bytes) - [ms]			
System	ETH	VPOST	VPRE	WIND
PrivateLab Node	0.720	3.746	3.709	2.967
Linux Box	0.335	3.631	3.699	2.586

	`tg` Average Jitter (2 Mbits/s - packet size : 250 Bytes) - [ms]			
System	ETH	VPOST	VPRE	WIND
PrivateLab Node	0.535	1.078	0.994	0.702
Linux Box	0.242	0.905	0.963	0.696